目　录

1.2 什么是数字图像处理

一幅图像可以定义为一个二维函数$f(x, y)$，其中x和y是空间坐标，而f在任意一对坐标(x, y)处的幅度称为该点处图像的亮度或灰度。当x, y和f的幅值都是有限的离散值时，称该图像为数字图像。数字图像处理就是用计算机处理数字图像。注意，数字图像是由有限数量的元素组成的，每个元素都有一个特殊的位置和数值。这些元素称为画素或像素。像素是广泛用于定义数字图像元素的术语。第2章中将正式地讨论这些定义。

视觉是我们感觉中最高级的，因此，图像在人类感知中起着最重要的作用并不令人奇怪。然而，人类的视觉被限制在电磁波谱的可视波段，而成像机器几乎覆盖了全部电磁波谱，其范围从伽马射线到无线电波。它们还可以在人类不常涉及的图像源所产生的图像上进行处理，包括超声波、电子显微镜和计算机产生的图像。这样，数字图像处理就包含了很宽的应用领域。

图像处理所涉及的领域到底有多广，作者们并无统一的见解。有时，人们将图像处理定义为其输入和输出均是图像的一个学科。但我们认为这存在局限性，并有点人为界定的意思。例如，在这种定义之下，计算图像的平均亮度这种简单任务将不被认为是图像处理操作。另外，存在像计算机视觉这样的领域，其最终目的是用计算机来模仿人类视觉，包括学习和推理，并根据视觉输入采取相应的行动。该领域本身是人工智能的一个分支，其目的是模仿人类智能。人工智能的研究领域从发展的意义上看还处于初始阶段，其进展要比预期的慢得多。图像分析领域（也称为图像理解）介于图像处理和计算机视觉之间。

图像处理和计算机视觉之间并没有明显的界限，但我们可通过考虑三种类型的计算机化处理来加以划分：低级、中级和高级处理。低级处理包括原始操作，如降低噪声的图像预处理、对比度增强和图像锐化。低级处理的特点是其输入与输出均为图像。图像的中级处理涉及诸如分割这样的任务，即把图像分为区域或对象，然后对对象进行描述，以便把它们简化为适合计算机处理的形式，并对单个对象进行分类（识别）。中级处理的特点是，其输入通常是图像，但输出则是从这些图像中提取的属性（如边缘、轮廓以及单个对象的特性）。最后，高级处理通过执行通常与人类视觉相关的感知函数，来对识别的对象进行总体确认。

基于前面的注释，我们可知图像处理和图像分析之间的重叠之处是图像中单个区域或对象的识别。这样，本书中所谓的数字图像处理就包含了其输入和输出都是图像的过程，从图像中提取特性的过程，以及对单个对象进行识别的过程。为说明这些概念，我们现在考虑文本的自动分析这一领域。该领域的图像获取过程，包括获取文本、预处理图像、提取（分割）个别字符、以适合计算机处理的形式描述字符以及识别这些个别字符，就在本书中所谓的数字图像处理范围之内。弄清这些内容后就了解了图像分析和计算机视觉的领域。正像我们所定义的那样，数字图像处理已成功用于许多领域，给人们带来了巨大的社会和经济价值。

1.3 MATLAB和图像处理工具箱的背景知识

MATLAB对于技术计算来说是一种高性能的语言。它以易于应用的环境集成了计算、可视化和编程，在该环境下，问题及其解以我们熟悉的数学表示法来表示。典型的应用包括如下方面：

- 数学和计算
- 算法开发
- 数据获取
- 建模、模拟和原型设计
- 数据分析、研究和可视化
- 科学和工程图形
- 应用开发，包括图像用户界面构建

MATLAB是一种交互式系统，其基本数据元素是并不要求确定维数的一个数组。这就允许人们用公式化方法求解许多技术计算问题，特别是涉及矩阵表示的问题。有时，MATLAB可调用使用C或Fortran这类非交互式语言所编写的程序。

MATLAB是matrix laboratory的缩写。MATLAB由LINPACK（Linear System Package）和EISPACK(Eigen System Package)项目开发，最初用于矩阵处理。今天，MATLAB已集成了LAPACK和BLAS库，并成为了矩阵计算的首选软件。

在高等院校中，对于数学、工程和科学理论中的入门课程和高级课程，MATLAB都是标准的计算工具。在工业领域，MATLAB对于研究、开发和分析也是首选的计算工具。MATLAB中补充了许多针对于特定应用的工具箱。图像处理工具箱是一个MATLAB函数（称为M函数或M文件）集，它扩展了MATLAB解决图像处理问题的能力。其他有时用于补充IPT的工具箱是信号处理、神经网络、模糊逻辑和小波工具箱。

MATLAB学生版MATLAB的一个极具特色的版本。学生版可以通过大学书店和MathWorks网站(www.mathworks.com)以较大的折扣购买到。包括图像处理工具箱在内的附加软件的学生版，也可通过类似的方式购买到。

1.4 本书涵盖的图像处理范围

本书中的每一章都包含有相关的MATLAB和IPT材料，以实现我们已讨论过的图像处理方法。当实现某种特殊方法的MATLAB或IPT函数不存在时，我们会开发一个新的函数并对之加以说明。正如前面提到的那样，本书中包括了所有的新函数。剩下的11章则涉及了如下内容。

第2章：基本原理。本章涵盖了MATLAB表示法、索引和编程概念的基础知识。这些内容是后续内容的基础。

第3章：亮度变换和空间滤波。本章详细讨论了使用MATLAB和IPT实现亮度变换函数的方法，并详细讨论与演示了线性和非线性滤波器。

第4章：频域处理。本章讨论了使用IPT函数计算傅里叶变换及其逆变换的方法，可视化傅里叶频谱的方法，以及在频域实现滤波的方法。此外，还讨论了由特定空间滤波器生成频域滤波器的方法。

第5章：图像复原。本章讨论了传统的线性复原方法，如维纳滤波。讨论并说明了非线性方法，如用于盲去卷积的Richardson-Lucy方法和最大似然估计。此外，还涉及了几何校正和配准。

第6章：彩色图像处理。本章讨论了伪彩色和全彩色图像处理，探讨了可用于数字图像处理的彩色模型，而附加彩色模型的实现则扩展了IPT的功能。本章还探讨了边缘检测和区域分割中彩色的应用。

第7章：小波。IPT目前没有任何小波变换。本章开发了与小波工具箱兼容的小波函数集，这些函数可帮助读者理解Gonzalez and Woods所著的书中讨论的所有小波变换概念。

第8章：图像压缩。工具箱中没有数据压缩函数。在这一章中，我们开发了一个可用于这一目的的函数集。

第9章：形态学图像处理。本章解释并说明了IPT中大量用于二值和灰度图像形态学图像处理的函数。

第10章：图像分割。本章解释并说明了用于图像分割的IPT函数集。此外，还开发了用于Hough变换处理和区域生长的新函数。

第11章：表示和描述。本章开发了几个用于对象表示和描述（包括链码和多边形表示）的新函数，还开发了几个用于对象描述的函数，如傅里叶描绘子、纹理和矩不变量。这些函数是对IPT中的区域特性函数集的补充。

第12章：对象识别。本章的重要特点之一是计算欧几里得距离和Mahalanobis距离的函数的有效实现。这些函数在模式匹配中扮演着重要角色。本章还包含有在MATLAB中如何操作符号串的广泛讨论。串操作和匹配在结构模式识别中很重要。

除前述内容外，本书还包括有三个附录。

附录A：包含了所有IPT和本书中开发的新图像处理函数，还包括了相应的MATLAB函数。这些有用的参考资料提供了工具箱和本书中所有函数的概览。

附录B：包含有在MATLAB中如何实现图形用户界面（GUI）的讨论。GUI是对本书材料的有用补充，因为它们简化了交互式函数的使用。

附录C：当在某章中解释一个新概念时，在该章的正文中会包含新函数的列表。除此之外，附录C中也包含了这些新函数的列表。过长的函数列表也包含在此处。某些函数的列表只在该附录中列出，目的在于不影响正文的连续阅读性。

1.5 本书的Web站点

本书特色之一是提供网站支持。网站地址为www.prenhall.com/gonzalezwoodseddins。该网站在如下领域为本书提供支持：

- 可下载的M文件，包括书中的所有M文件
- 培训
- 计划
- 授课材料
- 数据库链接，包括本书中的所有图像
- 书的更新
- 出版背景

该网站与Gonzalez and Woods所著的《数字图像处理》一书的网站集成在一起：

www.prenhall.com/gonzalezwoods

它为教学和研究提供了额外的支持。

1.6 MATLAB工作环境

本节简要介绍使用MATLAB的一些重要操作。

启动一个菜单，从菜单中可选择除执行命令外的各种选项。当在工作会话中试用各种命令时，这是很有用的特性。

1.6.2 使用MATLAB编辑器创建M文件

MATLAB编辑器既是用于创建M文件的文本编辑器，也是图形MATLAB调试器。编辑器可以自己以一个窗口出现出，或者以桌面上的子窗口出现。M文件用扩展符`.m`来表示，如`pixeldup.m`。MATLAB编辑器窗口有许多下拉菜单，用于保存、查看和调试文件。因为MATLAB编辑器可执行某些简单的检查，并且可用彩色区分各种编码元素，因此，在编写和编辑M函数时，应首选使用该文本编辑器。要打开该编辑器，可在命令窗口的提示符处键入`edit`命令。类似地，在提示符下键入`edit filename`，会在编辑器窗口打开M文件`filename.m`，编辑工作准备就绪。正像前面提到的那样，文件必须在当前目录中，或者在搜索路径的目录中。

1.6.3 获得帮助

获得在线帮助的主要方法是应用MATLAB帮助浏览器[①]，要打开帮助浏览器，可在桌面工具条上双击问号符（?），或在命令窗口提示符处键入`helpbrowser`。帮助浏览器是集成到MATLAB桌面的Web浏览器，它显示超文本标记语言（HTML）文档。帮助浏览器由两个面板组成，即用于寻找信息的帮助导航面板和用于查看信息的显示面板。导航面板上的自我解释标签用于执行搜索。例如，要得到特殊函数的帮助，可选择**Search**标签，并为**Search Type**选择**Function Name**，然后在**Search for**域中键入该函数的名称。在MATLAB会话开始后，最好打开帮助浏览器，以便在编码开发或其他MATLAB任务期间得到帮助。

获得某个函数的帮助的另一种方法是，在提示符处键入`doc`及该函数名。例如，若键入`doc format`，则会在帮助浏览器的显示面板中显示`format`函数的说明。若浏览器未打开，则该命令会打开浏览器。

M函数有两种可以由用户显示的信息类型。第一种信息类型称为H1行，它包含函数名和一行描述。第二种信息类型称为帮助文本块（详细内容将在2.10.1节中讨论）。在提示符处键入`help`及函数名，就会在命令窗口显示函数的H1行和帮助文本。有时，这种信息可能比帮助浏览器中的信息更新、更多，因为它是直接从相关M函数的文档中提取的。键入`lookfor`及一个关键字，会显示所有包含该关键字的H1行。在寻找特殊主题但又不知适用函数的名称时，该函数很有用。例如，在提示符处键入`lookfor edge`，会显示所有包含该关键字的H1行。因为H1行包含函数名，所以就有可能使用其他的帮助方法来查找指定的函数。在提示符处键入`lookfor edge -all`，会显示所有函数的H1行，而这些函数会在H1行或帮助文本块中包含单词`edge`。还会检测到包含字符`edge`的单词。例如，在H1行或帮助文本中包含单词`polyedge`的函数的H1行，也会显示出来。

除`lookfor`方法外，在使用前面描述的任何方法显示一个M函数的信息时，"帮助页"是MATLAB的常用术语。这里建议读者熟悉这些获取信息的方法。因为在后续章节中，我们常常只给出MATLAB和IPT函数的语法。这样做的原因有两个，一是本书的篇幅限制，二是防止讨论离题。在这些情况下，我们仅给出执行函数的语法。通过在线搜索方法，读者可以更详细地研究感兴趣的函数。

最后，1.3节中提到的MathWorks网站包含有丰富的帮助材料、他人开发的函数以及其他资源。

---

① 在本书中应用在线这一术语来指在本地计算机系统中可用的信息，而不是互联网上可用的信息。

1.6.4 保存和检索工作会话

在MATLAB中，保存和载入一个完整的工作会话或选取的工作空间变量有几种方法。最简单的方法如下。

为保存一个完整的工作空间，可简单地在工作空间浏览器窗口中的任何空白处右键单击，并在出现的菜单中选择**Save Workspace As**。这会打开一个目录窗口，该窗口允许命名文件及在系统中选择任何文件夹，并在文件夹中保存文件。然后，可简单地点击**Save**按钮。为了从工作空间保存一个所选的变量，可左键单击以选择该变量，然后在突出显示的区域右键单击。然后，在出现的菜单中选择**Save Selection As**。这将再次打开一个窗口，从中可选择一个文件夹来保存该变量。要选择多个变量，可Shift-单击或Control-单击，然后使用保存单个变量的过程。所有文件都以双精度二进制格式保存，扩展名为`.mat`。这些已保存的文件通常称为MAT文件。例如，名为mywork_2003_02_10的会话在保存时，将会出现MAT文件mywork_2003_02_10.mat。类似地，称为`final_image`的已保存图像（在工作空间中它是一个单变量）在保存时，将会以final_image.mat的形式出现。

要载入保存过的工作空间和/或变量，可在工作空间浏览器窗口的工具条上左键单击文件夹图标。这将打开一个窗口，这时可从中选择一个含有感兴趣MAT文件的文件夹。双击选中的MAT文件或选择**Open**按钮，可在工作空间浏览器窗口中恢复文件的内容。

在提示符处键人带有合适文件名及路径信息的`save`和`load`命令，也可以实现前几段描述的相同结果。这种方法虽然不太方便，但也有其优点。这里建议并鼓励读者使用帮助浏览器来更多地了解这两个函数的相关信息。

1.7 参考文献的组织方式

本书的所有参考文献都按作者和日期这种格式列出，如Soille[2003]。本书理论内容的大多数背景材料源于Gonzalez and Woods[2002]。非此情形时，会在讨论的适当位置给出新的参考文献。适用于所有章节的参考文献，如MATLAB手册和其他常用的参考文献，均标识在本书后面的“参考文献”中。

小结

除简单介绍表示法和基本的MATLAB工具外，本章还强调了在求解数字图像处理问题时理解原型环境的重要性。在后续各章中，我们将开始安排理解IPT函数所需的基础内容，并介绍一组贯穿全书的基本编程概念。第3章到第12章的内容跨越了很宽的主题，它们是数字图像处理应用的主流。尽管涉及的主题不同，但这些章节的讨论却遵循相同的基本主题，即演示并说明如何将MATLAB和IPT函数与新代码结合起来，以便求解宽泛的图像处理问题。

第2章　基本原理

前言

正如前一章所述，MATLAB为数字图像处理带来了一套广泛的函数，这些函数处理的是多维数组，而图像（二维数值数组）正是多维数组的一种特例。图像处理工具箱（IPT）是扩展MATLAB数值计算能力的函数集，这些函数与MATLAB语言的简洁表示，使得大量的图像处理操作可以按简洁明了的编码方式进行，从而为求解图像处理问题提供了一个理想的软件原型环境。在这一章中，我们将介绍MATLAB表示法的基本知识，讨论大量的IPT基本属性和函数，并介绍能进一步增强IPT的程序设计概念。因此，本章中的内容是后续大部分章节的内容的基础。

2.1　数字图像的表示

一幅图像可以被定义为一个二维函数$f(x, y)$，其中x和y是空间（平面）坐标，f在任何坐标点(x, y)处的振幅称为图像在该点的亮度。灰度是用来表示黑白图像亮度的一个术语，而彩色图像是由单个二维图像组合形成的。例如，在RGB彩色系统中，一幅彩色图像是由三幅独立的分量图像（红、绿、蓝）组成的。因此，许多为黑白图像处理开发的技术适用于彩色图像处理，方法是分别处理三幅独立的分量图像即可。彩色图像处理将在第6章详细讲解。

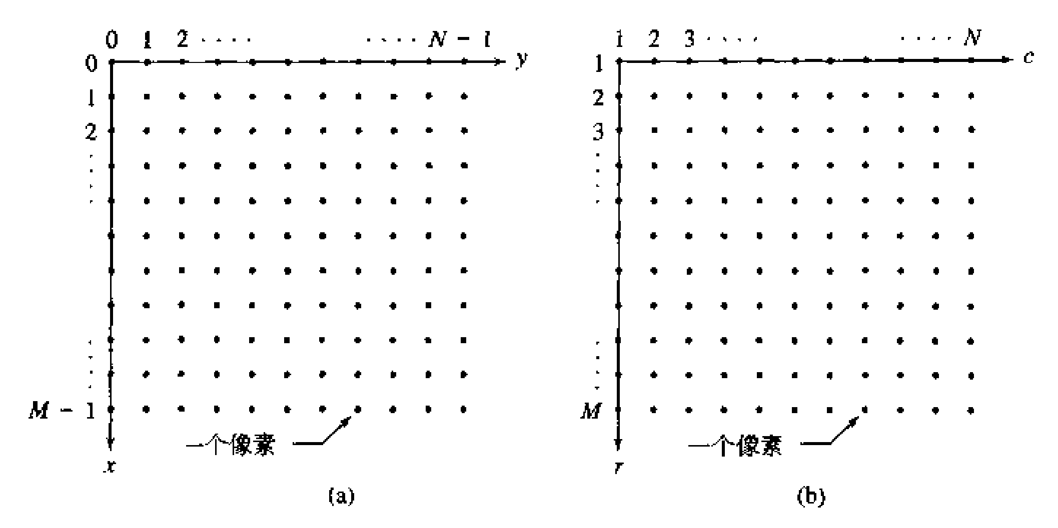
图像关于x和y坐标以及振幅连续。要将这样的一幅图像转换成数字形式，就要求数字化坐标和振幅。将坐标值数字化称为取样；将振幅数字化称为量化。因此，当f的x, y分量和振幅都是有限且离散的量时，称该图像为数字图像。

2.1.1　坐标约定

取样和量化的结果是一个实数矩阵。在本书中，我们使用两种主要的方法来表示数字图像。假设对一幅图像$f(x, y)$取样后，得到了一幅有着M行和N列的图像。我们称这幅图像的大小为$M \times N$。坐标(x, y)的值是离散量。为使符号表示清晰和方便，我们为这些离散坐标使用整数值。在很多图像处理书籍中，图像原点定义在$(x, y) = (0, 0)$处。沿图像第一行的下一坐标值为$(x, y) = (0, 1)$。注意，符号$(0, 1)$用来表示沿第一行的第二个取样，而不表示图像在取样时的实际物理坐标值。图2.1(a)显示了这种坐标约定。注意，x的范围是从0到$M - 1$的整数，y的范围是从0到$N - 1$的整数。

工具箱中用于表示数组的坐标约定与前段所述的坐标约定有两处不同。首先，工具箱使用(r, c)而不是(x, y)来表示行与列，但坐标顺序与前段所述的坐标顺序一致。在这种情况下，坐标元组(a, b)的第一个元素表示行，第二个元素表示列。另一区别是该坐标系统的原点在$(r, c) = (1, 1)$处。因此，r是从1到M的整数，c是从1到N的整数，如图2.1(b)所示。

IPT文档将图2.1(b)中的坐标称为像素坐标。IPT还采用另一种较少使用的坐标约定，称为空间坐标，这种坐标使用x来表示列，使用y来表示行。这与我们所用的变量x与y正好相反。在本书中，除少量例外外，我们将不使用IPT的空间坐标约定，但读者在IPT文档中一定会遇到这种术语。



(a) (b)

图2.1 所用的坐标约定：(a)多数图像处理书籍中所用的坐标约定；(b)图像处理工具箱中所用的坐标约定

2.1.2 图像的矩阵表示

由图2.1(a)所示的坐标系统和前述讨论，我们可以得到如下数字化图像函数的表示：

$$f(x,y)=\begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}$$

等式右边是由定义给出的一幅数字图像。该数组的每一个元素都称为像元、图元或像素。图像和像素这两个术语在本书后面的讨论中，将用来表示一幅数字图像及其元素。

一幅数字图像在MATLAB中可以很自然地表示成矩阵

$$\texttt{f}=\begin{bmatrix} \texttt{f(1,1)} & \texttt{f(1,2)} & \cdots & \texttt{f(1,N)} \\ \texttt{f(2,1)} & \texttt{f(2,2)} & \cdots & \texttt{f(2,N)} \\ \vdots & \vdots & & \vdots \\ \texttt{f(M,1)} & \texttt{f(M,2)} & \cdots & \texttt{f(M,N)} \end{bmatrix}$$

其中`f(1,1)`$=f(0,0)$，注意，等宽字体用来表示MATLAB的量。很明显，这两种表示是相同的，只是原点不同。符号`f(p,q)`表示位于p行和q列的元素。例如，`f(6,2)`是指矩阵f中位于第6行和第2列的元素。一般来说，我们分别用字母M和N来表示矩阵中的行与列。一个1 × N矩阵称为一个行向量，而一个M × 1矩阵称为一个列向量。一个1 × 1矩阵是一个标量。

在MATLAB中，矩阵以变量的形式来存储，名称诸如`A`,`a`,`RGB`,`real_array`等。变量必须以字母开头，且只能由字母、数字和下划线组成。如在前段中注释的那样，本书中的所有MATLAB量都用等宽字体来表示。此外，我们使用常见的斜体罗马字母来表示数学表达式，如$f(x,y)$。

2.2 读取图像

使用函数`imread`可以将图像读入MATLAB环境，`imread`的语法为

```
imread( 'filename' )
```

表 2.1 MATLAB 6.5 中的函数 imread 和 imwrite 所支持的一些常用图像/图形格式。早期版本支持这些格式的子集。所支持格式的完整清单请参阅在线帮助

格式名称	描述	可识别扩展符
TIFF	加标识的图像文件格式	.tif, .tiff
JPEG	联合图像专家组	.jpg, .jpeg
GIF	图形交换格式 *	.gif
BMP	Windows 位图	.bmp
PNG	可移植网络图形	.png
XWD	X Window 转储	.xwd

* imread 支持 GIF 格式，但 imwrite 不支持该格式。

其中，filename 是一个含有图像文件全名的字符串（包括任何可用的扩展名）。例如，命令行

```
>> f = imread('chestxray.jpg');
```

将JPEG图像（见表2.1）chestxray读入图像数组 f。注意，这里使用单引号(')来界定 filename 字符串。命令行结尾处的分号在MATLAB中用于取消输出。若命令行中未包含分号，则MATLAB会立即显示该行中指的运算的结果。在MATLAB命令行窗口中出现的提示符(>>)指明了命令行的开始（如图 1.1 所示）。

就像上面的这个命令行一样，当filename中不包含任何路径信息时，imread会从当前目录中寻找并读取图像文件（见 1.7.1 节）。若当前目录中没有所需要的文件，则它会尝试在 MATLAB 搜索路径中寻找该文件（见1.7.1节）。要想读取指定路径中的图像，最简单的办法就是在filename中输入完整的或相对的路径。例如，

```
>> f = imread('D: \myimages\chestxray.jpg');
```

从驱动器 D 上名为 myimages 的文件夹中读取图像文件 chestxray.jpg；而

```
>> f = imread('.\myimages\chestxray.jpg');
```

从当前的工作目录中名为myimages的子目录中读取图像文件 chestxray.jpg。MATLAB 桌面工具条上的当前目录窗口会显示MATLAB的当前工作路径，并提供一种非常简单的方法来手工改变当前的路径。表2.1列出了函数imread和imwrite所支持的常用图像/图形格式（函数imwrite将在 2.4 节中讨论）。

函数 size 可给出一幅图像的行数和列数：

```
>>size(f)
ans =
    1024  1024
```

在使用如下格式来自动确定一幅图像的大小时，该函数很有用：

```
>> [ M, N ] = size(f);
```

该语法将返回图像的行数（M）和列数（N）。

函数 whos 可以显示出一个数组的附加信息。例如，语句

```
>> whos f
```

会给出下列结果：

```
    Name        Size          Bytes        Class
    f           1024 x 1024   1048576      uint8 array
    Grand total is 1048576 elements using 1048576 bytes
```

结果中的uint8是指几种MATLAB数据类之一，详见2.5节的讨论。whos行结尾处的分号对结果没有影响，因此我们一般将其省略。

2.3 显示图像

在MATLAB桌面上图像一般使用函数imshow来显示，该函数的基本语法为

```
imshow(f, G)
```

其中，f是一个图像数组，G是显示该图像的灰度级数。若将G省略，则默认的灰度级数是256。语法

```
imshow(f, [low high])
```

会将所有小于或等于low的值都显示为黑色，所有大于或等于high的值都显示为白色。界于low和high之间的值将以默认的级数显示为中等亮度值。最后，语法

```
imshow(f, [ ])
```

可以将变量low设置为数组f的最小值，将变量high设置为数组f的最大值。函数imshow的这一形式在显示一幅动态范围较小的图像或既有正值又有负值的图像时非常有用。

函数pixval经常用来交互地显示单个像素的亮度值。该函数可以显示覆盖在图像上的光标。当光标随着鼠标在图像上移动时，光标所在位置的坐标和该点的亮度值会在该图形窗口的下方显示出来。处理彩色图像时，红、绿、蓝分量的坐标也会显示出来。若按下鼠标左键不放，则pixval将显示光标初始位置和当前位置间的欧几里得距离。

此处应注意的是，语法

```
pixval
```

会在上次显示的图像上显示光标。单击光标窗口上的X按钮可将其关闭。

例2.1 读入和显示图像

(a) 下列语句会从磁盘中读入一幅名为rose_512.tif的图像，提取该图像的基本信息，并使用imshow将其显示出来：

```
>> f = imread('rose_512.tif');
>> whos f
   Name        Size           Bytes       Class
   f           512 x 512      262144      uint8 array
   Grand total is 262144 elements using 262144 bytes
>> imshow(f)
```

由于imshow命令行结尾处的分号对结果无影响，所以一般将其省略。屏幕输出如图2.2所示。窗口左上角显示了图像编号。窗口上有各种下拉菜单和工具按钮，用于缩放、保存及输出显示窗口的内容等。特别地，在将结果打印或存盘之前，**Edit**菜单有对结果进行编辑或格式化的功能。

当用imshow显示另一幅图像g时，MATLAB会在屏幕上用新图像替换旧图像。为保持第一幅图像并同时显示第二幅图像，可以使用如下的figure函数：

```
>> figure, imshow(g)
```

使用语句

```
>> imwrite(f, 'patient10_run1', 'tif' )
```

或

```
>> imwrite(f, 'patient10_run1.tif' )
```

若 filename 中不包含路径信息，则 imwrite 会将文件保存到当前的工作目录中。

函数 imwrite 可以有其他的参数，具体取决于所选的文件格式。后续章节中的大部分工作都是处理 JPEG 或 TIFF 格式的图像，所以我们将注意力放在这两种格式上。

另一种常用但只适用于 JPEG 图像的函数是 imwrite，其语法为

```
imwrite(f, 'filename.jpg ', 'quality', q )
```

其中，q 是一个在 0 到 100 之间的整数（由于 JPEG 压缩，q 越小，图像的退化就越严重）。

例 2.2 使用函数 imfinfo 保存一幅图像

图 2.4(a)所示的图像 f 是由给定的化学过程得到的典型序列图像之一。我们期望定期将这些图像传送到中心站点，以便进行显示或自动检查。为降低存储量并减少传输时间，可在图像的外观质量不低于某个合理的级别时，尽量压缩图像。其中，“合理的”意味着没有可察觉的伪轮廓线。图 2.4(b)到图 2.4(f)示出了分别以 q = 50, 25, 15, 5 和 0 将图像 f 写到磁盘的结果（JPEG 格式）。例如，q = 25 时的语法结构为

```
>> imwrite(f, 'bubbles25.jpg', 'qyality', 25 )
```

以 q = 15 存储的图像［见图 2.4(d)］中几乎看不清伪轮廓线。当 q = 5 和 q = 0 时，伪轮廓线就变得非常明显。因此，消除伪边缘的一种可接受方案就是用参数 q = 25 来压缩图像。要了解所实现的压缩并获得图像文件的其他详细信息，可以使用 imfinfo 函数，其语法结构为

```
imfinfo filename
```

其中，filename 是存储在磁盘中的图像的全名。例如，

```
>> imfinfo bubbles25.jpg
```

会输出如下信息（注意，在这种情况下，有些域不包含信息）：

```
         Filename:  'bubbles25.jpg'
      FileModDate:  '04-Jan-2003 12:31:26'
         FileSize:  13849
           Format:  'jpg'
    FormatVersion:  ''
            Width:  714
           Height:  682
         BitDepth:  8
        ColorType:  'grayscale'
  FormatSignature:  ''
          Comment:  {}
```

其中，FileSize 以字节为单位。通过简单地使用 width 乘以 Height，再乘以 BitDepth，然后将结果除以 8，就可以计算出原图像中的字节数。计算的结果是 486 948。用这个结果除以 FileSize 就可以得到压缩比：(486 948/13 849) = 35.16。这一压缩比是在保持图像质量与应用要求一致的前提下得到的。除了在存储空间方面有明显的优势之外，这种压缩还可使得单位时间内传输的数据量大约是压缩前的 35 倍。

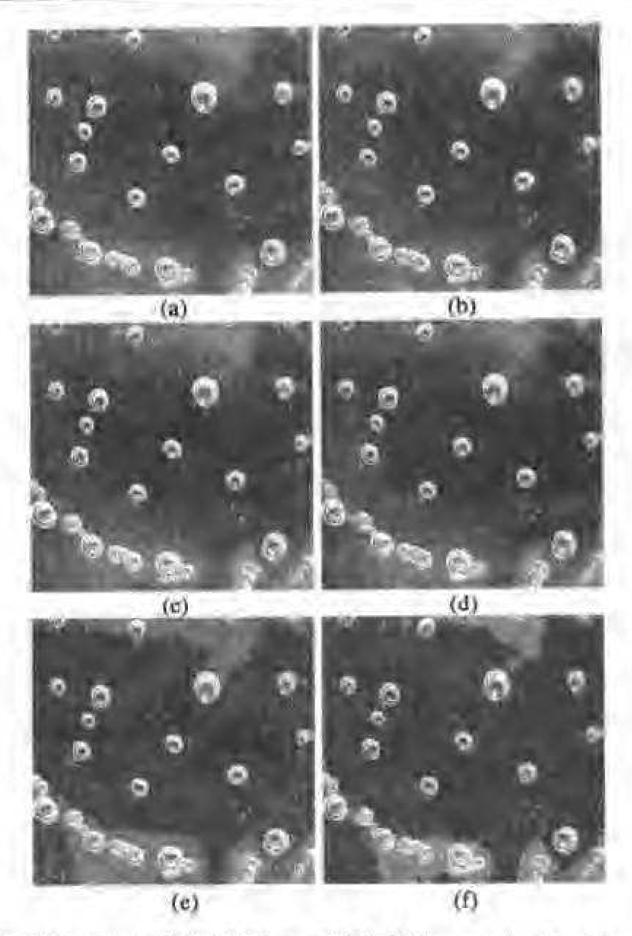


图 2.4 (a)原图像；(b)~(f)分别使用jpg质量参数q = 50,25,15,5,0的结果。q = 15时，伪轮廓并不明显[见图像(d)]，而在q = 5和0时则十分明显

由函数imfinfo显示的信息域可捕获至所谓的结构变量中，以便用于后续的计算。若以前面的这幅图像为例，并将相应的结构变量取名为K，则使用语法

```
>> K = imfinfo('bubbles25.jpg');
```

可将由命令imfinfo产生的所有信息存入变量K。由imfinfo产生的信息附加到了形式为K域的结构变量中。例如，图像的高度和宽度现在存储在结构域K.Height与K.Width中。下面我们通过计算图像bubbles25.jpg的压缩比来考虑使用结构变量K的一个示例：

```
>> K = imfinfo('bubbles25.jpg');
>> image_bytes = K.Width*K.Height*K.BitDepth/8;
>> compressed_bytes = K.FileSize;
>> compression_ratio = image_bytes/compressed_bytes
compression_ratio =
    35.1612
```

注意，imfinfo有两种不同的用法。第一种用法是在命令行键入imfinfo bubbles25.jpg，其结果是在屏幕上显示出信息。第二种用法是键入K = imfinfo('bubbles25.jpg')，其结果是把由imfinfo产生的信息存入K。调用imfinfo的这两种不同方法是命令-函数二元性的一个例子，这个重要的概念在MATLAB的在线文档中有更加详细的解释。

函数imwrite的另一种常用但只适用于tif图像的语法为

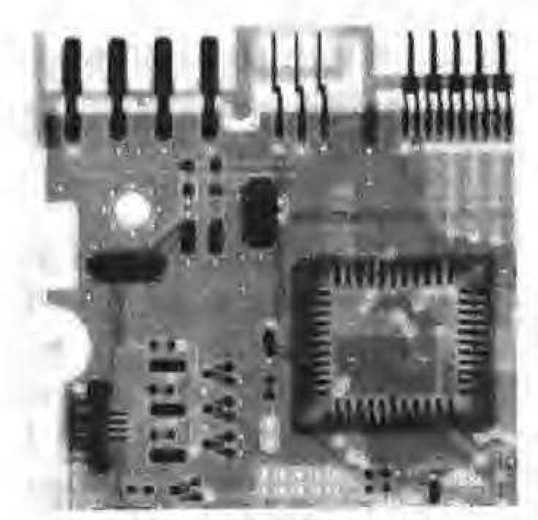
```
imwrite(g, 'filename.tif', 'compression', 'parameter', ...
                           'resolution', [colres rowres])
```

其中，'parameter'可以是如下主要的值之一：'none'表示无压缩；'packbits'表示比特包压缩(非二值图像的默认参数)；'ccitt'表示ccitt压缩(二值图像的默认参数)。1×2矩阵[colres rowres]包含两个整数，分别以每单位中的点数给出图像的列分辨率和行分辨率(默认值为[72 72])。例如，若一幅图像的大小以英寸来表示，则colres是垂直方向上每英寸的点(像素)数(dpi)，而rowres是水平方向上每英寸的点数。使用标量res来指定分辨率与使用[res res]是等价的。

例2.3　使用imwrite的参数

图2.5(a)是检查一块电路板的质量时产生的一幅8比特X光图像。其格式为jpg，分辨率为200 dpi。图像大小为450 × 450像素，即其尺寸为2.25 × 2.25英寸。我们想把这幅图像以tif格式存储为无压缩的名为sf的图像，并且想将图像的尺寸减小到1.5 × 1.5英寸，但其像素数仍保持为450 × 450。下列语句可产生我们所希望的结果：

```
>> imwrite(f,'sf.tif','compression','none','resolution', ...
                                              [300 300])
```



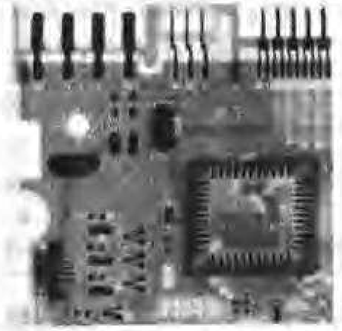


图2.5　更改dpi分辨率并同时保持像素数不变时的效果：(a)分辨率为200 dpi且大小为450 × 450的图像(大小为2.25 × 2.25英寸)；(b)大小为450 × 450(1.5 × 1.5英寸)且分辨率为300 dpi的同一幅图像(原图像由Lixi公司提供)

向量[colres rowres]的值由分辨率200 dpi乘以压缩比2.25/1.5得到，即300 dpi。可以用下列语句来代替手工计算：

```
>> res = round(200*2.25/1.5);
>> imwrite(f, 'sf.tif', 'compression', 'none' ,'resolution', res)
```

其中，函数round会将其参量舍入为最接近的整数。注意，这些命令并不改变像素数，而只改变图像的大小。原图像的像素数为450 × 450，分辨率为200 dpi、尺寸为2.25 × 2.25英寸。

分辨率为 300 dpi 的新图像与原图像完全一致，只是其 450 × 450 像素分布在 1.5 × 1.5 英寸的区域内。这样的过程在打印文档时控制图像的大小而不牺牲其分辨率是非常有用的。

通常，有必要将图像按它们在MATLAB桌面上显示的那样输出到磁盘中。对于下一章将要讲到的绘图，更应如此。图形窗口的内容可以按两种方法输出到磁盘。第一种方法是在图形窗口的**File**下拉菜单（见图2.2）中选择**Export**。使用这一选项，用户可以选择保存路径、文件名以及文件格式。使用print 命令可以获得更多的输出参数：

```
print -fno -dfileformat -rresno filename
```

其中，*no*是感兴趣的图形窗口的图形编号，*fileformat* 是表 2.1 中的一种文件格式，*resno* 是单位为 dpi 的分辨率，filename 是我们希望为文件指定的文件名。例如，要将图 2.2 所示图形窗口的内容以 300 dpi 的分辨率输出到一个名为 hi_res_rose 的 tif 文件中，可以键入

```
>> print -f1 -dtiff -r300 hi_res_rose
```

该命令会把文件 hi_res_rose 发送到当前目录中。

若在命令行中简单地键入print，则MATLAB 会（使用默认的打印机）打印上一个显示的图形窗口的内容。也可以指定 print 的其他选项，如某个指定的打印设备。

2.5 数据类

虽然我们处理的是整数坐标，但 MATLAB 中的像素值本身并不是整数。表 2.2 中列出了 MATLAB 和 IPT 为表示像素值所支持的各种数据类①。表中的前 8 项称为数值数据类，第 9 项称为字符类，最后一项称为逻辑数据类。

MATLAB 中所有的数值计算都可用 double 类来进行，所以它也是图像处理应用中最常使用的数据类。uint8 数据类也是一种频繁使用的数据类，尤其是在从存储设备中读取数据时，因为 8 比特图像是实际中最常用的图像。logical 类和使用较少的 uint16 类构成了本书集中讨论的基础数据类。但是，很多 IPT 函数都支持表 2.2 中列出的所有数据类。double 数据类需要使用 8 个字节来表示一个数字，而uint8和int8只需要1个字节，uint16和int16需要2个字节，uint32、int32 和 single 则需要 4 个字节。char 数据类用来表示 Unicode 字符。一个字符串就是一个 1 × n 字符矩阵。logical 类矩阵中每个元素的取值只能是 0 和 1，并且每个元素都用 1 字节存储在存储器中。逻辑矩阵的创建可通过函数 logical（见 2.6.2 节）或相关的运算符来实现（见 2.10.2 节）。

表 2.2 数据类。前 8 项称为数值数据类，第 9 项称为字符类，最后一项称为逻辑数据类

名称	描述
double	双精度浮点数，范围为 $-10^{308} \sim 10^{308}$（8 比特每像素）
unit8	无符号 8 比特整数，范围为 [0, 255]（1 比特每像素）
uint16	无符号 16 比特整数，范围为 [0, 65 535]（2 比特每像素）
uint32	无符号 32 比特整数，范围为 [0, 4 294 967 295]（4 比特每像素）
int8	有符号 8 比特整数，范围为 [-128, 127]（1 比特每像素）
int16	有符号 16 比特整数，范围为 [-32 768, 32 767]（2 比特每像素）
int32	有符号 32 比特整数，范围为 [-2 147 483 648, 2 147 483 647]（4 比特每像素）
single	单精度浮点数，范围为 $-10^{38} \sim 10^{38}$，（4 比特每像素）
char	字符（2 比特每像素）。
logical	值为 0 或 1（1 比特每像素）

① MATLAB 文献经常互换地使用术语“数据类”和“数据类型”。本书中，如 2.6 节讨论的那样，我们把“类型”术语留给图像使用。

2.6 图像类型

工具箱支持以下4种图像类型：

- 亮度图像（Intensity images）
- 二值图像（Binary images）
- 索引图像（Indexed images）
- RGB图像（RGB images）

大多数单色图像的处理运算是通过二值图像或亮度图像来进行的，所以我们首先重点研究这两种图像。索引图像和RGB图像将在第6章中讨论。

2.6.1 亮度图像

一幅亮度图像是一个数据矩阵，其归一化的取值表示亮度。若亮度图像的像素都是`uint8`类或`uint16`类，则它们的整数值范围分别是[0, 255]和[0, 65 535]。若图像是`double`类，则像素的取值就是浮点数。规定双精度型归一化亮度图像的取值范围是[0, 1]。

2.6.2 二值图像

二值图像在MATLAB中具有非常特殊的意义。一幅二值图像是一个取值只有0和1的逻辑数组。因而，一个取值只包含0和1的`uint8`类数组，在MATLAB中并不认为是二值图像。使用`logical`函数可以把数值数组转换为二值数组。因此，若`A`是一个由0和1构成的数值数组，则可使用如下语句创建一个逻辑数组`B`：

```
B = logical(A)
```

若`A`中含有除了0和1之外的其他元素，则使用`logical`函数就可以将所有非零的量变换为逻辑1，而将所有的0值变换为逻辑0。使用关系和逻辑运算符也可以创建逻辑数组（见2.10.2节）。

要测试一个数组是否为逻辑数组，可以使用函数`islogical`：

```
islogical(c)
```

若`c`是逻辑数组，则该函数返回1；否则，返回0。使用2.7.1节所讨论的数据类转换函数，可将逻辑数组转换为数值数组。

2.6.3 术语注释

前两节用了大量的笔墨来阐明术语“数据类”和“图像类型”。通常，在我们提到一幅图像时，是指一幅“`data_class` image_type图像”，其中的`data_class`是表2.2中的类之一，而image_type则是本节开始时定义的图像类型之一。因此，一幅图像的特性是由数据类和图像类型这两者来表征的。例如，“`uint8`亮度图像”表示一幅像素都是`uint8`数据类的亮度图像。工具箱中的有些函数支持所有的数据类，而有些函数只支持特殊的数据类。例如，前面提到的二值图像中的像素只能是`logical`数据类。

2.7 数据类与图像类型间的转换

在IPT应用中，数据类与图像类型间的转换是非常频繁的操作。在数据类间转换时，记住表2.2中详细列出的每种数据类的取值范围是很重要的。

2.7.1 数据类间的转换

数据类间的转换相当直接。通用的语法为

```
B = data_class_name(A)
```

其中，data_class_name 可以是表 2.2 中第一列的任何一项。例如，假设 A 是一个 uint8 类数组，则命令 B = double(A) 会产生一个双精度数组 B。这种转换贯穿全书，因为 MATLAB 希望数值计算中的所有操作数都是双精度浮点数。假设 C 是一个取值范围为[0, 255]（很有可能包含小数）的 double 类数组，则命令 D = uint8(C)可将其转换为一个 uint8 类数组。

若一个 double 类数组包含有区间[0, 255]之外的值，则在使用上述方法将其转换成 uint8 类数组时，MATLAB 会将所有小于 0 的值转换为 0，所有大于 255 的值转换为 255，而在 0 和 255 之间的值将全部舍去小数部分转换为整数。因此，在将 double 类数组转换成 uint8 类数组之前，有必要先对其进行适当的缩放，以使其元素的取值尽量在区间[0, 255]内。如2.6.2节中提到的那样，在将任何数值数据类转换为 logical 类时，数组中的所有非 0 值将转换为逻辑 1，0 值将转换为逻辑 0。

2.7.2 图像类和类型间的转换

工具箱中提供了执行必要缩放的函数（见表 2.3），以在图像类和类型间进行转换。函数 im2uint8 可以检测出输入的数据类，并进行所有必要的缩放，以便使工具箱能将这些数据识别为有效的图像数据。例如，考虑下面这个 double 类 2 × 2 图像 f，它可以是中间计算的结果：

```
f =
      -0.5   0.5
      0.75   1.5
```

执行转换

```
>> g = im2uint8(f)
```

得出结果

```
g =
       0      128
     191      255
```

可以看出，函数 im2uint8 将输入中所有小于 0 的值设置为 0，而将输入中所有大于 1 的值设置为 255，再将所有的其他值乘以 255。将得到的结果四舍五入为最接近的整数后，就完成了转换。注意，im2uint8 的舍入行为与前一节中讨论的数据类转换函数 uint8 是不一样的，后者只是简单地将小数部分全部舍去。

表 2.3 IPT 中用于进行图像类和类型间的转换的函数。适用于彩色图像的转换请参阅表 6.3

名称	将输入转换为	有效的输入图像数据类
im2uint8	uint8	logical, uint8, uint16 和 double
im2uint16	uint16	logical, uint8, uint16 和 double
mat2gray	double，范围为[0, 1]	double
im2double	double	logical, uint8, uint16 和 double
im2bw	logical	uint8, uint16 和 double

要把一个 double 类的任意数组转换成取值范围为[0, 1]的归一化 double 类数组，可以通过函数 mat2gray 完成，其基本语法为

转换为一幅这样的二值图像，即原图像中的 1 和 2 变为 0，其余两个值变为 1。首先，我们将原图像的取值范围变换为区间[0, 1]：

```
>> g = mat2gray(f)
g =
         0    0.3333
    0.6667    1.0000
```

然后，我们使用阈值 0.6 将其转换为二值图像：

```
>> gb = im2bw(g, 0.6)
gb =
     0   0
     1   1
```

正如在 2.5 节中提到的那样，我们可以使用关系运算符（见 2.10.2 节）直接生成一个二值数组。这样，我们就可使用如下语句得到相同的结果：

```
>> gb = f > 2
gb =
     0   0
     1   1
```

我们可以使用函数 islogical 将 gb 是一个逻辑数组的事实存储为一个变量（如 gbv）：

```
>> gbv = islogical(gb)
gbv =
     1
```

(b) 假设我们现在要将 gb 转换为一个值为 0 和 1 的 double 类数值数组。实现的方法如下：

```
>> gbd = im2double(gb)
gbd =
     0   0
     1   1
```

若 gb 是一个 uint8 类数值数组，则对其使用函数 im2double 可以得到取值为

```
     0         0
0.0039    0.0039
```

的数组，因为函数 im2double 会将所有的元素都除以 255。这在前面的转换中都没有出现过，因为那时函数 im2double 检测到输入是一个值只能为 0 和 1 的逻辑数组。若输入是一个 uint8 类数值数组，则在保持其值为 0 和 1 的前提下要将它转换为 double 类数组，可以使用如下语句完成：

```
>> gbd = double(gb)
gbd =
     0   0
     1   1
```

最后要指出的是，MATLAB 支持嵌套语句。因此，上述对图像 f 产生相同结果的所有操作，可只用单行语句

```
>> gbd = im2double(im2bw(mat2gray(f), 0.6));
```

来完成，或者使用这些函数的部分组合来完成。当然，在这种情况下，整个过程也可以用一条更简单的命令实现：

```
>> gbd = double(f > 2);
```

这又一次证明了MATLAB语言的简洁性。

2.8 数组索引

MATLAB支持大量功能强大的索引方案，这些索引方案不仅简化了数组操作，而且提高了程序的运行效率。本节将讨论并举例说明一维和二维(向量和矩阵)的基本索引。后面的讨论将根据需要介绍更复杂的技术。

2.8.1 向量索引

如我们在2.1.2节中讨论的那样，维数为$1 \times N$的数组称为行向量。行向量中元素的存取是使用一维索引进行的。因此，v(1)是向量v的第一个元素，v(2)是第二个元素，依次类推。MATLAB中向量的元素使用方括号括起，并由空格或逗号隔开。例如，

```
>> v = [1   3   5   7   9]
v =
     1   3   5   7   9
>> v(2)
ans =
     3
```

使用转置运算符(.')可将行向量转换为列向量：

```
>> w = v.'
w =
    1
    3
    5
    7
    9
```

要存取元素的数据块，我们可使用MATLAB的冒号。例如，要存取v的前三个元素，可使用语句

```
>> v(1:3)
ans =
     1   3   5
```

类似地，也可以使用如下语句存取第二个到第四个元素：

```
>> v(2:4)
ans =
     3   5   7
```

或使用如下语句存取第三个到最后一个元素：

```
>> v(3:end)
ans =
     5   7   9
```

其中，end 表示向量中的最后一个元素。若 v 是一个向量，则

```
>> v(:)
```

产生一个列向量，而语句

```
>> v(1:end)
```

产生一个行向量。

索引并不限于连续的元素。例如，

```
>> v(1:2:end)
ans =
    1   5   9
```

注意，符号 1:2:end 表示索引从 1 开始计数，步长为 2，直到最后一个元素时停止。步长也可以为负：

```
>> v(end:-2:1)
ans =
    9   5   1
```

这时，索引从最后一个元素开始计数，步长为 -2，直到第一个元素时停止。

函数 linspace 的语法为

```
x = linspace(a, b, n)
```

该语句产生一个含有 n 个元素的行向量 x，这 n 个元素之间线性地隔开并且包含 a 与 b。后续章节会在一些地方使用到该函数。

一个向量也可用做另一个向量的索引。例如，我们可以使用如下命令挑出向量 v 的第一个、第四个和第五个元素：

```
>> v([1  4  5])
ans =
    1   7   9
```

就像下一节所示的那样，使用一个向量作为另一个向量的索引这种功能，在矩阵索引中起着非常重要的作用。

2.8.2 矩阵索引

在 MATLAB 中，矩阵可以很方便地用一列被方括号括起并用分号隔开的行向量表示。例如，

```
>> A = [1  2  3;  4  5  6;  7  8  9]
```

显示了 3 × 3 矩阵

```
A =
    1   2   3
    4   5   6
    7   8   9
```

注意，此处分号的作用，与前面提到的取消命令行的输出或在一行中写入多条命令时所用分号的作用是不同的。

从矩阵中选取元素和从向量中选取元素是一样的，但我们现在需要两个索引：一个用于确定行位置，另一个用于确定相应的列位置。例如，要提取第 2 行第 3 列的元素，可使用语句

```
5
8
3
6
9
```

使用单个下标存储A可直接对该列索引。例如，A(3)存取的是列中的第三个值，即数字7；A(8)存取的是第8个值，即数字6，如此等等。当我们使用这个列符号时，表示我们正简单地对所有元素A(1:end)寻址。这种类型的索引是为优化程序而使循环向量化的基本成分，详见2.10.4节中的讨论。

例2.5 使用数组索引进行简单的图像操作

图2.6(a)是一幅大小为 1024 × 1024 的 uint8 类亮度图像 f。图2.6(b)是该图像经过如下语句操作后垂直翻转的图像：

```
>> fp = f(end:-1:1, :);
```

图2.6(c)所示的图像是图像(a)中的一部分，由以下语句获得：

```
>> fc = f(257:768,  257:768);
```

类似地，图2.6(d)是使用如下语句进行二次取样后得到的图像：

```
>> fs = f(1:2:end, 1:2:end);
```

最后，图2.6(e)显示了通过图2.6(a)中部的一条水平扫描线，它是使用如下语句得到的：

```
>> plot(f(512, :))
```

函数 plot 将在3.3.1节中详细讨论。

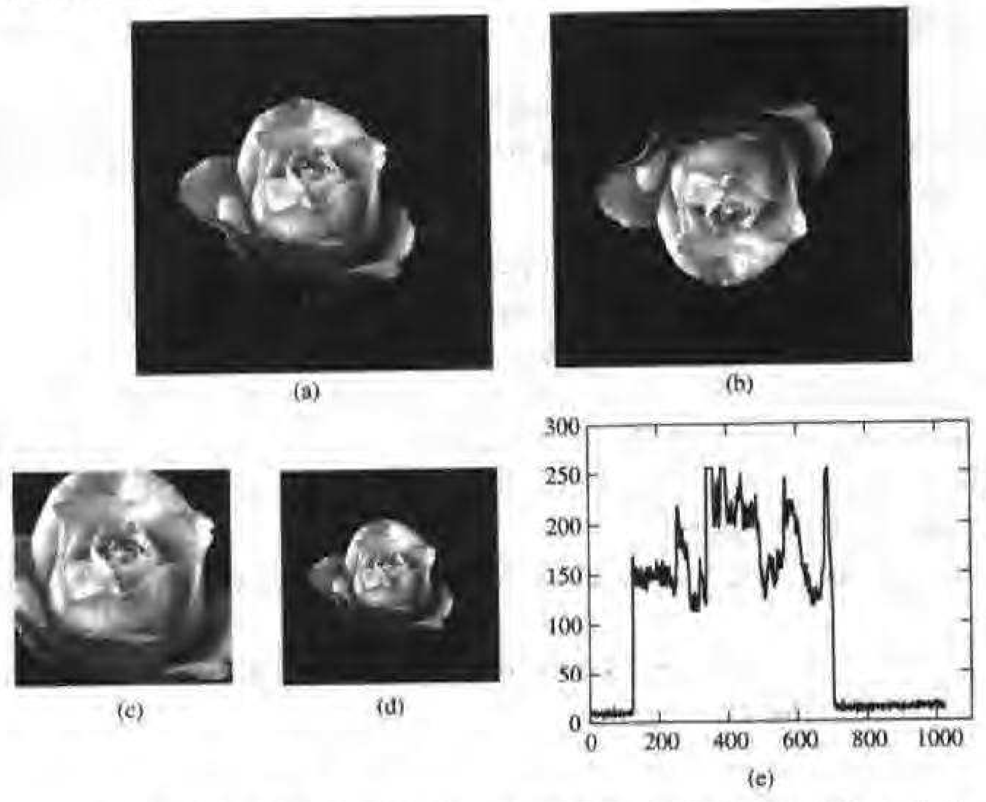


图2.6 使用数组索引得到的结果：(a)原图像；(b)垂直翻转的图像；(c)裁剪后的图像；(d)二次取样后的图像；(e)通过图(a)中间的水平扫描线

2.8.3 选择数组的维数

本书中频繁地使用了形如

```
operation(A, dim)
```

的操作，operation 表示 MATLAB 中的一种可用操作，A 是一个数组，dim 是一个标量。例如，假设 A 是一个大小为 $M \times N$ 的数组。命令

```
>> k = size(A, 1);
```

沿 A 的第一个维数（在 MATLAB 中定义为垂直方向）给出 A 的大小。换言之，该命令给出 A 的行数。类似地，数组的第二个维数为水平方向，所以语句 size(A,2) 给出 A 的列数。单一维数是任意维数 dim，且 size(A,dim) = 1。使用这些概念，我们可将例 2.5 中的最后一条命令写为

```
>> plot(f(size(f, 1)/2, : ))
```

MATLAB 并不限制数组的维数，因此，以任何一个维数来提取某个数组的分量就成为了一个重要的特性。大部分情况下，我们只处理二维数组，但有时（如处理彩色图像或者多谱段图像时）需要将图像“堆叠”到第三维或更高维上去。这些内容将会在第 6 章、第 11 章和第 12 章中详细解释。函数 ndims 的语法为

```
d = ndims(A)
```

它将给出数组 A 的维数。函数 ndims 返回的值不会小于 2，因为即使是标量，我们也认为它有两个维数，这时的标量是大小为 1 × 1 的数组。

2.9 一些重要的标准数组

在开发期间，用一些简单的图像数组来检验算法并测试函数的语法常常是很有用的。本节将介绍 7 种数组生成函数，这些函数将在后面的章节中用到。在下面的任何一个函数中，若只包含一个参量，则结果将是一个方阵。

- zeros(M,N) 生成一个大小为 M × N 的 double 类矩阵，其元素均为 0。
- ones(M,N) 生或一个大小为 M × N 的 double 类矩阵，其元素均为 1。
- ture(M,N) 生成一个大小为 M × N 的 logical 类矩阵，其元素为 1。
- false(M,N) 生成一个大小为 M × N 的 logical 类矩阵，其元素为 0。
- magic(M) 生成一个大小为 M × N 的“魔术方阵”。在该方阵中，每一行中的元素之和、每一列中的元素之和以及主对角线中的元素之和均相等。魔术方阵可用于测试目的，因为它们易于生成，且其元素均为整数。
- rand(M,N) 生成一个大小为 M × N 的矩阵，矩阵中的元素都是在区间[0, 1]中均匀分布的随机数。
- randn(M,N) 生成一个大小为 M × N 的矩阵，矩阵中的元素是正态分布（如高斯分布）的随机数，随机数的均值为 0，方差为 1。

例如，

```
>> A = 5*ones(3, 3)
A =
```

```
     5     5     5
     5     5     5
     5     5     5
>> magic(3)
ans =
     8     1     6
     3     5     7
     4     9     2
>> B = rand(2, 4)
B =
     0.2311    0.4860    0.7621    0.0185
     0.6068    0.8913    0.4565    0.8214
```

2.10 M函数编程简介

图像处理工具箱最强大的特征之一是其对MATLAB编程环境的透明访问。MATLAB函数编程非常灵活，并且非常容易学习，随着我们学习的深入，这一特性将会很快呈现。

2.10.1 M文件

MATLAB中的M文件可以是简单执行一系列MATLAB语句的脚本，也可以是接受变量并产生一个或多个输出的函数。本节重点介绍M文件函数。这些函数将MATLAB和IPT的功能扩展到了寻址用户定义的特定应用。

M文件由文本编辑器创建，并以filename.m形式的文件名存储，如average.m和filter.m。M文件函数的组成部分为

- 函数定义行
- H1行
- 帮助文本
- 函数体
- 命令

函数定义行的形式为

```
function [outputs] = name(inputs)
```

例如，计算两幅图像的和与积（两个不同的输出）的函数的形式为

```
function [s, p] = sumprod(f, g)
```

其中f和g是输入图像，s是和图像，p是积图像。名称sumprod可任意定义，但字function必须出现在左侧，如上所示。注意，输出变量必须位于方括号内，而输入变量必须位于圆括号内。若函数只有单个输出变量，则也可不使用括号而直接列出。若函数没有输出，则只需要使用字function，而无须括号或等号。函数名必须以字母开头，后面可以跟字母、数字、下划线的任意组合，但不允许有空格。MATLAB可以识别长达63个字符的函数名，超过此长度的字符将被忽略。

函数可以在命令提示符处调用；例如，

```
>> [s, p] = sumprod(f, g);
```

也可以用做其他函数的元素，此时的函数就成为了子函数。正如前一段提到的那样，若输出只有一个变量，则也可不使用括号，例如

```
>> y = sum(x);
```

H1 行是第一个文本行。它是单个注释行，其前面为函数定义行。H1 行与函数定义行间无空行或前导空格。H1 行的一个例子为

```
% SUMPROD Computes the sum and product of two images.
```

如我们在 1.7.3 节中提到的那样，当用户在 MATLAB 提示符处键入

```
>> help function_name
```

时，H1 行是最先出现的文本。此外，该节还提到键入 lookfor keyword 会显示出所有含有字符串 keyword 的 H1 行。该行提供了关于 M 文件的重要摘要信息，所以应尽可能地描述它。

帮助文本是紧跟在 H1 行后面的文本块，二者之间无空行。帮助文本用来为函数提供注释或在线帮助。当用户在提示符处键入 help function_name 时，MATLAB 会显示函数定义行和第一个非注释（可执行或空白的）行之间的全部注释行。帮助系统会忽略帮助文本块后的任何注释行。

函数体包含了所有执行计算并给输出变量赋值的 MATLAB 代码。MATLAB 代码的几个例子将在本章后面给出。

符号“%”后面非 H1 行或帮助文本的所有行可看做是函数注释行，它们不是帮助文本的一部分。代码行的末尾可附加注释。

M 文件可以使用任何文本编辑器创建和编辑，并以扩展名 .m 保存到指定的目录下，一般会保存在 MATLAB 搜索路径中。创建 M 文件的另一种方法是在提示符处使用 edit 函数。例如，若文件存在于 MATLAB 搜索路径的目录中或者在当前目录中，则键入

```
>> edit sumprod
```

会打开文件 sumprod.m 并进行编辑。若找不到该文件，MATLAB 会为用户提供创建该文件的选项。如 1.7.2 节中提到的那样，MATLAB 编辑器窗口有很多下拉菜单，可以完成诸如保存文件、查看文件以及调试文件等任务。由于文本编辑器可以执行一些简单的检查并使用不同的颜色来区分各种代码元素，所以建议用户在写 M 文件或编辑 M 文件时使用该文本编辑器。

2.10.2 运算符

MATLAB 运算符可以分为以下三种主要类别：

- 执行数值计算的算术运算符
- 在数量上比较操作数的关系运算符
- 执行函数 AND、OR 和 NOT 的逻辑运算符

本节的剩余部分将讨论这些运算符。

算术运算符

MATLAB 有两种不同类型的算术运算符。矩阵算术运算符由线性代数的规则定义。数组算术运算符可以逐个元素地执行，并且可以用于多维数组。句点（圆点）字符（.）用来区分数组运算符与矩阵运算符。例如，A*B 表示传统意义上的矩阵乘法，而 A.*B 则表示数组乘法，这种乘法的

乘积是与A和B大小相同的数组，其每个元素都是A和B中相应元素的乘积。换言之，若C=A.*B，则C(I,J) = A(I,J)*B(I,J)。由于对加法和减法来说，矩阵运算和数组运算是相同的，所以不使用字符对.+和.-。

在编写如B = A这样的表达式时，MATLAB将做一个B等于A的“记录”，但实际上并不将数据复制到B，除非在程序中A的内容以后有了变化。这一点很重要，因为使用不同的变量来“存储”相同的内容有时可以增强代码的清晰性和可读性。因此，在编写MATLAB代码时，一定要记住MATLAB不复制信息这一功能。表2.4列出了MATLAB的算术运算符，其中A与B是矩阵或数组，a与b是标量。所有的操作数可以是实数或复数。若操作数是标量，则数组运算符中的小圆点可以省略。记住，图像是等价于矩阵的二维数组，因此，表中的所有运算符都适用于图像。

表2.4 数组和矩阵算术运算符。涉及这些运算符的计算可使用运算符本身实现，如A + B，或使用所示的MALAB函数实现，如plus(A,B)。为数组显示的示例使用矩阵来简化表示，但它们可很容易地扩展到高维情形

运算符	名称	MATLAB函数	注释和示例
+	数组和矩阵加	plus(A,B)	a+b,A+B或a+A
-	数组和矩阵减	minus(A,B)	a-b,A-B,A-a或a-A
.*	数组乘	times(A,B)	C=A.*B,C(I,J)=A(I,J)*B(I,J)
*	矩阵乘	mtimes(A,B)	A*B,即标准矩阵乘；或a*A，即一个标量乘以A的所有元素
./	数组右除	rdivide(A,B)	C=A./B,C(I,J)=A(I,J)/B(I,J)
.\	数组左除	ldivide(A,B)	C=A.\B,C(I,J)=B(I,J)/A(I,J)
/	矩阵右除	mrdivide(A,B)	A/B与A*inv(B)大致相同，具体取决于计算精度
\	矩阵左除	mldivide(A,B)	A\B与inv(A)*B大致相同，具体取决于计算精度
.^	数组求幂	power(A,B)	若C=A.^B，则C(I,J)=A(I,J)^B(I,J)
^	矩阵求幂	mpower(A,B)	该运算符的讨论请参阅在线帮助
.'	向量和矩阵转置	transpose(A)	A.'。标准的向量和矩阵转置
'	向量和矩阵复共轭转置	ctranspose(A)	A'。标准的向量和矩阵共轭转置。当A是实数时，A.'=A'
+	一元加	uplus(A)	+A与0+A相同
-	一元减	uminus(A)	-A与0-A或-1*A相同
:	冒号		详见2.8节的讨论

表2.5 IPT支持的图像算术函数

函数	描述
imadd	两幅图像相加或把常数加到图像
imsubtract	两幅图像相减或从图像中减去常数
immultiply	两幅图像相乘，其中相乘是在相应的像素对间进行的；或图像乘以一个常数
imdivide	两幅图像相除，其中相除是在相应的像素对间进行的，或图像除以一个常数
imabsdiff	计算两幅图像间的绝对差
imcomplement	对图像求补，参见3.2.1节
imlincomb	计算两幅或多幅图像的线性组合，示例请参阅5.3.1节

工具箱支持表2.5中列出的图像算术函数。尽管这些函数可以使用MATLAB算术运算符直接实现，但使用IPT函数的优点在于它们支持整数数据类，而等同的MATLAB数学运算符要求输入是double类数据。

后面的例2.6用到了函数max和min。前者的语法形式有如下几种：

```
C = max(A)
C = max(A, B)
C = max(A, [ ], dim)
[C, I] = max(...)
```

在第一种形式中，若A是一个向量，则max(A)返回其最大元素；若A是一个矩阵，则max(A)将A的列作为向量来处理，并返回一个包含了每列最大值的行向量。在第二种形式中，max(A,B)返回一个与A和B大小相同的数组，该数组由A与B中最大的元素组成。在第三种形式中，max(A, [ ],dim)返回沿标量dim指定的A的维度上的最大元素。例如，max(A,[ ],1)产生的是A的第一维（行）上的最大值。最后，[C, I] = max(...)会找出A的最大值的索引，并将它们返回到输出向量I中。若有几个相等的最大值，则返回找到的第一个最大值的索引。圆点表示使用的语法是前三种形式的右侧这一。函数min的语法形式与函数max的语法形式类似。

例2.6 算术运算符与函数max和min的示例

假设我们要编写一个称为fgprod的M函数，该函数的功能是将两幅输入图像相乘，并输出图像的乘积、乘积的最大值与最小值以及一幅归一化的乘积图像（其取值范围为[0, 1]）。使用文本编辑器，我们可写出期望的函数，如下所示：

```
function [p, pmax, pmin, pn] = improd(f, g)
%IMPROD Computes the product of two images.
%  [P, PMAX, PMIN, PN] = IMPROD(F, G)① outputs the element-by-
%  element product of two input images, F and G, the product
%  maximum and minimum values, and a normalized product array with
%  values in the range [0, 1]. The input images must be of the same
%  size. They can be of class uint8, unit16, or double. The outputs
%  are of class double.

fd = double(f);
gd = double(g);
p = fd.*gd;
pmax = max(p(:));
pmin = min(p(:));
pn = mat2gray(p);
```

注意，输入图像是使用函数double而不是函数im2double来转换为double类图像的。因为，若输入是uint8类图像，则函数im2double会将它们转换到范围[0, 1]。假定我们想将原始值的乘积放在p中。为得到一个取值范围为[0, 1]的归一化数组pn，我们可以使用函数mat2gray。还要注意单个冒号索引的使用，详见2.8节中的讨论。

假设f = [1 2;3 4]，g = [1 2;2 1]。在提示符处键入上述函数可以得到如下输出：

```
>> [p, pmax, pmin, pn] = improd(f, g)
p =
     1     4
     6     4
pmax =
```

① 在MATLAB文献中，在指函数名和参量时，在H1行和帮助文本中常常使用大写字母，以避免程序名或变量与通常的解释性文本相混淆。

```
    6
pmin =
    1
pn =
          0    0.6000
     1.0000    0.6000
```

在提示符处键入help improd可以得到如下输出：

```
>> help improd

IMPROD Computes the product of two images.
  [P, PMAX, PMIN, PN] = IMPROD(F, G) outputs the element-by-
  element product of two input images, F and G, the product
  maximum and minimum values, and a normalized product array with
  values in the range [0, 1]. The input images must be of the same
  size.  They can be of class uint8, unit16, or double. The outputs
  are of class double.
```

关系运算符

表2.6列出了MATLAB的关系运算符。这些运算符可以对相同维数的数组中的相应元素进行比较，或逐个元素地进行比较。

例2.7 关系运算符

尽管关系运算符的关键应用是在2.10.3节讨论的流控制中（如在if语句中），但这里还是简单地演示一下这些运算符是怎样直接用于数组的。考虑下面的输入和输出序列：

```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
    1    2    3
    4    5    6
    7    8    9
>> B = [0 2 4; 3 5 6; 3 4 9]
B =
    0    2    4
    3    5    6
    3    4    9
>> A == B
ans =
    0    1    0
    0    1    1
    0    0    1
```

可以看到，运算符A==B生成一个与A和B维数相同的逻辑数组，当A与B的相应元素匹配时，新数组中的相应位置取1，其余位置取0。考虑另一个例子，语句

```
>> A >= B
ans =
    1    1    0
    1    1    1
    1    1    1
```

生成一个逻辑数组，当A的元素大于或等于B的相应元素时，新数组中的相应位置取1，其余位置取0。

表 2.6 关系运算符

运算符	名称
<	小于
<=	小于等于
>	大于
>=	大于等于
==	等于
~=	不等于

对于向量和矩形数组，两个操作数必须有相同的维数，或者其中一个操作数是标量。此时，MATLAB 将这个标量与另一个操作数的每一个元素相比较，产生一个与操作数大小相同的逻辑数组，在满足指定关系的位置取 1，其余位置取 0。若两个操作数都是标量，则当指定关系满足时结果为 1，否则为 0。

逻辑运算符与函数

表2.7列出了MATLAB的逻辑运算符，且下面的例子说明了其中一些运算符的性质。不同于逻辑运算符的通用描述，表 2.7 中的运算符既能作用于逻辑数据又能作用于数值数据。在所有的逻辑测试中，MATLAB 将逻辑 1 或非零数值量作为 `true` 来处理，将逻辑 0 和数值 0 作为 `false` 来处理。例如，当两个操作数都为逻辑 1 或非零数值时，两个操作数 AND 运算的结果为 1；当两个操作数中的任何一个是逻辑 0 或数值 0 时，或两个操作数都为逻辑 0 或数值 0 时，AND 运算的结果为 0。

表 2.7 逻辑运算符

运算符	名称
&	AND（与）
|	OR（或）
~	NOT（非）

例 2.8 逻辑运算符

考虑下面两个数值数组的 AND 运算：

```
>> A = [1 2 0; 0 4 5];
>> B = [1 -2 3; 0 1 1];
>> A & B

ans =
     1     1     0
     0     1     1
```

可以看出，AND运算符产生一个与输入数组大小相同的逻辑数组，并在两个输入数组相应操作数都为非零值的位置取 1，其他位置取 0。像前面一样，每次运算都在输入数组的相应位置的一对操作数上进行。

OR 运算符的使用方式与 AND 运算符的使用方式类似。当两个操作数中任何一个为逻辑 1 或非零数值时，或两个操作数都为逻辑1或非零数字时，OR运算表达式的值为 `ture`；否则为 `false`。NOT 运算符只作用于单个输入。逻辑上，若一个操作数为 `ture`，则 NOT 运算符会将其转换为 `false`。当将 NOT 作用于数值数据时，所有非零操作数将变为 0，零操作数变为 1。

MATLAB 也支持表 2.8 中总结的逻辑函数。在编程中，函数 `all` 与 `any` 是非常有用的。

例2.9 逻辑函数

考虑简单的数组A=[1 2 3;4 5 6]与B=[0 -1 1;0 0 2]。将表2.8中的函数作用于这两个数组，可得到如下结果：

```
>> xor(A, B)

ans =
     1     0     0
     1     1     0
>> all(A)

ans =
     1     1     1
>> any(A)

ans =
     1     1     1
>> all(B)

ans =
     0     0     1
>> any(B)

ans =
     0     1     1
```

表2.8 逻辑函数

函数	注释
xor（异或）	若两个操作数逻辑上不同，则函数xor返回1；否则，返回0
all	若一个向量中的所有元素都非零，则函数all返回1；否则，返回0
any	若一个向量中的任何元素都非零，则函数any返回1；否则，返回0。该函数在矩阵中按列操作

仔细观察函数all与any是如何作用于A和B的列的。例如，由all(B)产生的向量的前两个元素为0，因为B的前两列中每一列都至少含有一个0；最后一个元素为1，因为B的最后一列的所有元素都非零。

除了表2.8中列出的函数，MATLAB还提供了很多其他的函数，这些函数可以检测到特殊条件或特殊值的存在并返回逻辑的结果。表2.9列出了一些这样的函数。其中一小部分涉及到了本章前面提到的一些术语和概念（如2.6.2节中的islogical函数）；其他的函数将在后面的讨论中用到。记住，表2.9中列出的函数在条件满足时将返回逻辑1，否则将返回逻辑0。当变量是数组时，表2.9中的一些函数会产生一个与输入变量大小相同的数组，在满足函数条件的位置取逻辑1，其他位置取逻辑0。例如，假设A=[1 2;3 1/0]，函数isfinite(A)会返回矩阵[1 1;1 0]，该矩阵中的0（假）表示A的最后一个元素不是有限值。

表2.9 在变量为ture或false时,根据数值或条件返回逻辑1或0的某些函数

函数	描述
iscell(C)	若C是单元数组，则为真
iscellstr(s)	若s是字符串单元数组，则为真
ischar(s)	若s是字符串，则为真
isempty(A)	若A是空数组[]，则为真
isequal(A,B)	若A和B有相同的元素和维数，则为真
isfield(S,'name')	若'name'是结构S的一个域，则为真
isfinite(A)	若数组A的元素有限，则为真

```
   error('The dimensions of the input cannot exceed 2.')
end
% Compute the average
av = sum(A(:))/length(A(:));
```

注意，输入已使用A(:)转换为一个一维数组。一般来说，length(A)返回数组A的最长维的大小。在该例中，由于A(:)是一个向量，所以length(A)给出了A中的元素个数，从而无须测试输入是一个向量还是一个二维数组。另一种直接获得数组中的元素个数的方法是使用函数numel，其语法为

```
n = numel(A)
```

因此，若A是一幅图像，则numel(A)将给出它的像素数。使用这个函数时，前一个程序中最后一条可执行的语句就变为

```
av = sum(A(:))/numel(A);
```

最后要注意的是，函数error将终止程序的执行并且输出括号内的信息（引号不能省略）。

for

如表2.11所示，一个for循环按指定的次数执行一组语句。其语法为

```
for index = start:increment:end
    statements
end
```

可以嵌套两个或更多的for循环，如下所示：

```
for index1 = start1:increment1:end
   statements1
   for index2 = start2:increment2:end
      statements2
   end
   additional loop1 statements
end
```

例如，下面的循环将执行11次：

```
count = 0;
for k = 0:0.1:1
   count = count + 1;
end
```

若省略了循环增量，则其默认为1。循环增量也可以为负，如k = 0:-1:-10。注意，每个for行的结尾无须加分号。MATLAB能够自动地停止打印循环索引的值。如我们将在2.10.4节中讨论的那样，利用所谓的向量化代码代替for循环，程序的执行速度会有明显的提高。

例2.11 使用for循环将多幅图像写入文件

例2.2中比较了几幅使用不同JPEG质量值的图像。这里，我们将给出使用for循环来把这些文件写入磁盘的方法。假设存在一幅图像f，我们想将其写为一系列JPEG文件，这些文件用范围在0至100、增量为5的质量因子表示。进而，假设我们想将这些JPEG文件用series_xxx.jpg来命名，其中xxx是品质因子。使用下面的for循环，我们可完成这项任务：

```
for q = 0:5:100
   filename = sprintf('series_%3d.jpg', q);
   imwrite(f, filename, 'quality', q);
end
```

在此例中，函数 sprintf 的语法为

```
s = sprintf('characters1%ndcharacters2', q)
```

它会将格式化的数据写为一个字符串s。在这种语法形式中，characters1和characters2是字符串，%nd表示一个n位十进制数（由q决定）。在此例中，characters1代表series_，n的值是3，characters2代表.jpg，q的值在循环中指定。

while

只要控制循环的表达式为true，while循环就会执行一组语句。其语法为

```
while expression
   statements
end
```

与for循环类似，while循环也可以嵌套使用：

```
while expression1
   statements1
   while expression2
      statements2
   end
   additional loop1 statements
end
```

例如，如下嵌套的while循环会在a和b均降至0时终止：

```
a = 10;
b = 5;
while a
   a = a - 1;
   while b
      b = b - 1;
   end
end
```

注意，为了控制循环，我们采用了MATLAB以逻辑形式来表达数值的习惯，即对非零数用true表示，0用false表示。换言之，只要a与b非零，则while a和while b均为真。

与for循环类似，利用向量化代码（见2.10.4节）代替while循环，程序的执行速度会有明显的提高。

break

正如其名称所示的那样，break终止for或while循环的执行。遇到break语句时，程序会继续执行循环外的下一条语句。在嵌套循环中，break语句仅退出包含它的最内层循环。

continue

continue语句将控制传递给for或while循环的下一次迭代，而跳过循环体中的任何其他语句。在嵌套循环中，continue语句会将控制传递给包含该语句的循环的下一次迭代。

switch

这是一个基于不同类型的输入来控制 M 函数的流的选择语句。其语法为

```
switch switch_expression
   case case_expression
      statement(s)
   case {case_expression1, case_expression2,...}
      statement(s)
   otherwise
      statement(s)
end
```

switch构造基于变量或表达式的值来执行语句组。关键词case和otherwise用于描述语句组。仅执行首先匹配的 case[①]。通常必须有 end来匹配 switch语句。当同一个 case语句中存在多个表达式时，则必须使用大花括号。例如，假设存在一个M函数，该函数接收一幅图像 f，并将其转换为一个指定的类，称该类为newclass。该转换只对三种图像类有效，即uint8类图像、uint16类图像和double类图像。下列代码片断执行我们所期望的转换，并且会在输入图像不是所允许转换的类时输出错误信息：

```
switch newclass
   case 'uint8'
      g = im2uint8(f);
   case 'uint16'
      g = im2uint16(f);
   case 'double'
      g = im2double(f);
   otherwise
      error('Unknown or improper image class.')
end
```

全书广泛使用了 switch 构造。

例 2.12　从给定图像中提取子图像

在本例中，我们将基于for循环来编写一个M函数，以便从一幅图像中提取一幅矩形子图像。正如下一节将要讨论的那样，尽管我们可以使用单条MATLAB语句来完成提取，但使用这个例子可在稍后比较 for 循环和向量化代码的运行速度。函数的输入是一幅图像、所提取子图像的大小（行数和列数）以及子图像左上角的坐标。记住，MATLAB 中图像的原点在(1, 1)处，详见 2.1.1 节的讨论。

```
function s = subim(f, m, n, rx, cy)
%SUBIM Extracts a subimage, s, from a given image, f.
%  The subimage is of size m-by-n, and the coordinates
%  of its top, left corner are (rx, cy).

s = zeros(m, n);
rowhigh = rx + m - 1;
colhigh = cy + n - 1;
xcount = 0;
```

① 与C语言中的switch构造不同，MATLAB中的switch不会“失败”。换言之，switch仅执行第一个匹配的情形，而不执行后续的匹配情形。因此，未使用break语句。

```
for r = rx:rowhigh
   xcount = xcount + 1;
   ycount = 0;
   for c = cy:colhigh
      ycount = ycount + 1;
      s(xcount, ycount) = f(r, c);
   end
end
```

下一节将给出上述代码的更有效的实现。作为练习，读者可以使用while循环代替for循环来完成上面这个程序。

2.10.4 代码优化

正如1.3节中详细讨论的那样，MATLAB是一种专门为数组运算而设计的编程语言。只要可能，利用这一优点就可加快计算速度。在这一节中，我们将讨论两种优化M ATLAB代码的方法，即向量化循环和预分配数组。

向量化循环

向量化意味着简单地将将for循环和while循环转换为等价的向量或矩阵运算。运算除了明显地变得简短之外，向量化不仅可加速计算，还可增强代码的可读性。尽管多维向量化有时很难公式化，但在图像处理中使用的向量化形式通常是相当直接的。

我们从一个简单的例子开始。假设我们要生成一个一维函数，该函数的形式为

$$f(x) = A\sin(x/2\pi)$$

其中，$x = 0, 1, 2, \cdots, M-1$。实现该计算的for循环为

```
for x = 1:M    % Array indices in MATLAB cannot be 0.
   f(x) = A *sin((x - 1)/(2*pi)) ;
end
```

但在向量化后，代码的效率会更高；换言之，通过充分利用MATLAB索引，上述代码可简化为

```
x = 0:M - 1;
f = A*sin(x/(2*pi));
```

正像这个简单的例子显示的那样，一维索引通常很简单。当将被评估的函数有两个变量时，优化的索引可能会复杂一些。MATLAB使用函数meshgrid来实现二维函数的评估，该函数的语法为

```
[C, R] = meshgrid(c, r)
```

该函数将由行向量c和r指定的域变换为数组C和R，这两个数组能用来评估有着两个变量的函数和三维表面图（注意，在meshgrid的输入和输出中，列总是首先列出）。

输出数组C的行是向量c的副本，输出数组R的列是向量r的副本。例如，假设我们想形成一个二维函数，该函数的元素是坐标变量x和y的值的平方和，其中x = 0,1,2和y = 0,1。向量r由坐标的行分量构成：r = [0,1,2]；类似地，向量c由坐标的列分量构成：c = [0 1]（注意此处的r和c均为行向量）。将这两个向量代入meshgrid可得如下数组：

```
>> [C, R]= meshgrid(c, r)
```

```
C =
     0    1
     0    1
     0    1
R =
     0    0
     1    1
     2    2
```

我们感兴趣的这个函数将如下实现：

```
>> h = R.^2 + C.^2
```

这将得到如下结果：

```
h =
     0    1
     1    2
     4    5
```

注意，h的维数是length(r) × length(c)。还要注意h(1,1) = R(1,1)^2 + C(1,1)^2。这样，MATLAB会自动处理索引h。当坐标中涉及0时，这会是混乱之源，因为本书和手册中反复强调MATLAB数组不能有0索引。正如这个简单的例子所示，形成h时，MATLAB为进行计算使用了R和C中的内容。h，R和C的索引由1开始。下一个例子将进一步演示索引方案的功能。

例2.13 向量化的可计算优点示例以及计时函数tic和toc的介绍

在本例中，我们将编写一个M函数，以便比较使用for循环和向量化代码的如下二维图像函数的实现情况：

$$f(x, y) = A\sin(u_0x + v_0y)$$

其中$x = 0, 1, 2, \cdots, M-1$和$y = 0, 1, 2, \cdots, N-1$。我们还将介绍计时函数tic和toc。

函数的输入是A, u_0, v_0, M和N。期望的输出是由两种方法生成的图像（它们应该是相同的），以及使用for循环实现此函数所用的时间与使用向量化代码实现此函数所用的时间之比。具体的解决方案如下所示：

```
function [rt, f, g] = twodsin(A, u0, v0, M, N)
%TWODSIN Compares for loops vs. vectorization.
%  The comparison is based on implementing the function
%  f(x, y) = Asin(u0x + v0y) for x = 0, 1, 2,..., M-1 and
%  y = 0, 1, 2, ..., N - 1. The inputs to the function are
%  M and N and the constants in the function.

%  First implement using for loops.

tic    % Start timing.

for r = 1:M
   u0x = u0*(r - 1);
   for c = 1:N
      v0y = v0*(c - 1);
      f(r, c) = A*sin(u0x + v0y);
   end
end
t1 = toc;    % End timing.
```

```
% Now implement using vectorization. Call the image g.
tic  % Start timing.
r = 0:M - 1;
c = 0:N - 1;
[C, R] = meshgrid(c, r);
g = A*sin(u0*R + v0*C);
t2 = toc;   % End timing.
% Compute the ratio of the two times.
rt = t1/(t2 + eps); % Use eps in case t2 is close to 0.
```

在MATLAB的提示符处运行函数

```
>> [rt, f, g] = twodsin(1, 1/(4*pi), 1/(4*pi), 512, 512);
```

可得rt的如下值：

```
>> rt
rt =
    34.2520
```

使用函数mat2gray可将生成的图像（f和g相同）转换为可视形式：

```
>> g = mat2gray(g);
```

使用函数imshow可显示可视图像：

```
>> imshow(g)
```

图2.7显示了所得的结果。

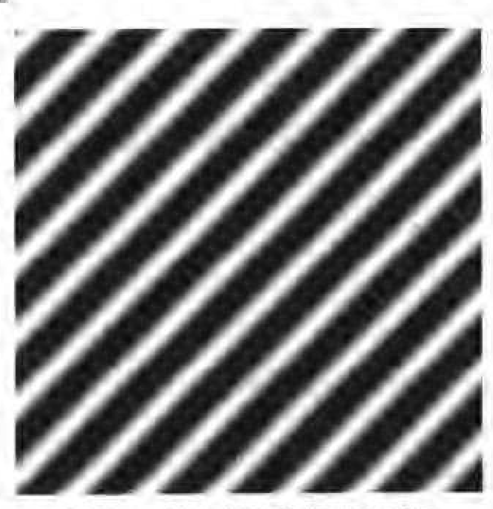


图2.7 例2.13生成的正弦图像

例2.13中的向量化代码运行时要比基于for循环的实现快30倍。当程序运行时间较长时，这个显著的计算优势就显示出其深远的意义。例如，若M和N较大且向量化的程序需要运行2分钟，则基于for循环完成相同的任务需要近1小时的时间。这时就要尽量向量化程序，尤其是在程序中使用到子程序时。

先前关于向量化的讨论集中在图像坐标的计算上。通常，我们所感兴趣的是从一幅已知图像中提取或处理一块区域。若所要提取的区域为矩形且包括该矩形中的所有像素，则提取这种区域的程

序的向量化非常简单，因而向量化经常用在此类操作中。要提取大小为m × n且其左上角坐标为(rx,cy)的区域s，可使用如下所示的基本的向量化代码：

```
rowhigh = rx + m - 1;
colhigh = cy + n - 1;
s = f(rx:rowhigh, cy:colhigh);
```

其中，f为将从其中提取区域的图像。利用for循环实现相同的任务已在例2.12中给出。在例2.13中，用两种方法实现并对其计时表明，向量化代码的运行速度要比基于for循环的代码的运行速度快1000倍左右。

预分配数组

加快代码执行时间的另一种方法是在程序中预分配数组的大小。在处理数值或逻辑数组时，预分配只是简单地创建有着适当维数的数组，数组的元素均为0。例如，若我们正在处理两幅大小均为1024 × 1024像素的图像f和g，则预分配由如下语句构成：

```
>> f = zeros(1024); g = zeros(1024);
```

大处理大数组时，预分配也可帮助我们减少存储器碎片。动态存储器的分配和去分配会使得存储器出现碎片。实际的结果是在计算过程中可能会有足够空间的可用物理存储器，但可能没有足够的连续空间来容纳一个较大的变量。预分配通过在计算开始时就允许MATLAB为大数据构造保留足够的存储空间，来阻止无连续空间的情形出现。

2.10.5 交互式I/O

通常，我们希望编写交互式M函数，以便既能够向用户显示信息和指令，又能够接收来自键盘的输入信息。在这一节中，我们将为编写这样的函数打下基础。

函数disp用来在屏幕上显示信息。其语法为

```
disp( argument )
```

若argument是一个数组，则disp显示数组的内容。若argument是一个文本串，则disp显示串中的字符。例如，

```
>> A = [1 2; 3 4];
>> disp(A)
    1    2
    3    4
>> sc = 'Digital Image Processing.';
>> disp(sc)
Digital Image Processing.
>> disp('This is another way to display text.')
This is another way to display text.
```

注意，只显示argument的内容，而不显示如ans =这样的字（即通过省略命令行结尾处的分号而显示一个变量的值时，我们经常在屏幕上看到的字）。

函数input用于将数据输入到M函数。其基本语法为

```
t = input('message')
```

该函数输出message中包含的内容并等待来自用户的输入，随后是一个回车，并将输入保存到t中。输入信息可以是单个数字、字符串（用单引号括起）、向量（用方括号括起，其中的内容用空格或逗号分开）、矩阵（用方括号括起，行与行之间用分号隔开），或者任何有效的MATLAB数据结构。其语法

```
t = input('message', 's')
```

输出message的内容并接收一个字符串，字符串的内容可用逗号或空格隔开。由于这种语法允许多个独立输入，所以非常灵活。若要求输入为数字，则可使用函数str2num将串的元素（作为字符处理）转换为double类数字。其中，函数str2num的语法为

```
n = str2num(t)
```

例如，

```
>> t = input('Enter your data: ', 's')
Enter your data: 1, 2, 4
t =
    1 2 4
>> class(t)
ans =
    char
>> size(t)
ans =
    1    5
>> n = str2num(t)
n =
    1    2    4
>> size(n)
ans =
    1    3
>> class(n)
ans =
    double
```

其中，t是一个大小为1 × 5的字符数组（三个数字和两个空格）；n是一个大小为1 × 3的向量，其数字均为double类。

若输入中既有字符又有数字，则可以利用MATLAB中的串处理函数之一。现在的讨论所要用到的是strread函数，其语法为

```
[a, b, c, ...] = strread(cstr, 'format', 'param', 'value')
```

该函数使用指定的format和param/value的组合，从字符串cstr中读取数据。在本章中，我们感兴趣的格式是%f和%q，分别表示浮点数和字符串。对于param项，我们使用delimiter，以表明format中识别的项将由value中指定的字符分隔（一般为逗号或空格）。例如，假设我们有一个串

```
>> t = '12.6, x2y, z';
```

要将该输入的元素读入三个变量a,b和c，代码可写为

```
>> [a, b, c] = strread(t, '%f%q%q', 'delimiter', ',')
```

```
a =
    12.6000
b =
    'x2y'
c =
    'z'
```

输出 a 为 double 类；围绕输出 x2y 和 z 的单引号表示 b 和 c 是单元数组，这将在下一节中详细讨论。令

```
>> d = char(b)
d =
    x2y
```

我们可将 b 转换为字符数组，对 c 也可进行类似的操作。格式化串中的元素数（及顺序）必须匹配左侧期望的输出变量的个数与类型。在这种情况下，我们期望三个输入：一个浮点数和两个字符串。

函数 strcmp 用于比较串。例如，假设我们有一个 M 函数 g = imnorm(f,param)，该函数读取一幅图像 f 和一个参数 param。参数 param 有两种形式，即 'norm1' 和 'norm255'。首先，f 将被缩放到范围 [0, 1] 内；其次，它将缩放到范围 [0, 255] 内。在两种情况下的输出都为 double 类。下列代码片断可完成所需要的归一化：

```
f = double(f);
f = f - min(f(:));
f = f./max(f(:));
if strcmp(param, 'norm1')
    g = f;
elseif strcmp(param, 'norm255')
    g = 255*f;
else
    error('Unknown value of param.')
end
```

若 param 中指定的值不是 'norm1' 或 'norm255'，则会出现错误。此外，若用于归一化因子的不全是小写字符，也会出现错误。我们可以修改该函数，以便可接受小写或大写字符，方法是使用函数 lower 将任何输入转换为小写字符，如下所示：

```
param = lower(param)
```

同样，若代码使用大写字母，则我们也可以使用函数 upper 将任意输入字符串转换为大写形式：

```
param = upper(param)
```

2.10.6 单元数组与结构简介

在处理混合变量（如字符与数字）时，可以充分利用单元数组。MATLAB 中的单元数组是一个多维数组，其元素是其他数组元素的副本。例如，单元数组

```
c = {'gauss', [1 0; 0 1], 3}
```

包含了三个元素：一个字符串、一个大小为 2 × 2 的矩阵和一个标量（注意，应使用大括号将数组括起）。要选择一个单元数组的内容，可用大括号括起一个整数地址。在这种情况下，我们会得到如下结果：

第3章　亮度变换与空间滤波

前言

术语“空间域”指的是图像平面本身，在空间域内处理图像的方法是直接对图像的像素进行处理。在本章中，我们着重讨论两种重要的空间域处理方法，即亮度（或灰度级）变换与空间滤波。后一种方法有时称为邻域处理或空间卷积。在下一节中，我们将举例说明在MATLAB中使用这两种方法的处理技术。为使讨论的主题一致，本章中的大部分例题都与图像增强有关。这是介绍空间处理的很好方法，因为增强技术对于初学者来说是高度直观且容易接受的。纵观全书，这些技术还广泛应用于数字图像处理的许多其他领域。

3.1　背景知识

像前一段指出的那样，空间域技术直接对图像的像素进行操作。本章中讨论的空间域处理由表达式

$$g(x, y) = T[f(x, y)]$$

表示，其中$f(x, y)$为输入图像，$g(x, y)$为输出（处理后的）图像，T是对图像f进行处理的操作符，定义在点(x, y)的指定邻域内。此外，T还可以对一组图像进行处理，例如为降低噪声而让K幅图像相加。

定义点(x, y)的空间邻近区域的主要方法是，使用中心位于(x, y)的正方形或长方形区域，如图3.1所示。此区域的中心从原点（如左上角）开始逐像素点移动，在移动的同时，该区域会包含不同的邻域。T应用于每个位置(x, y)，以便在该位置得到输出图像g。在计算(x, y)处的g值时，只使用该邻域的像素。

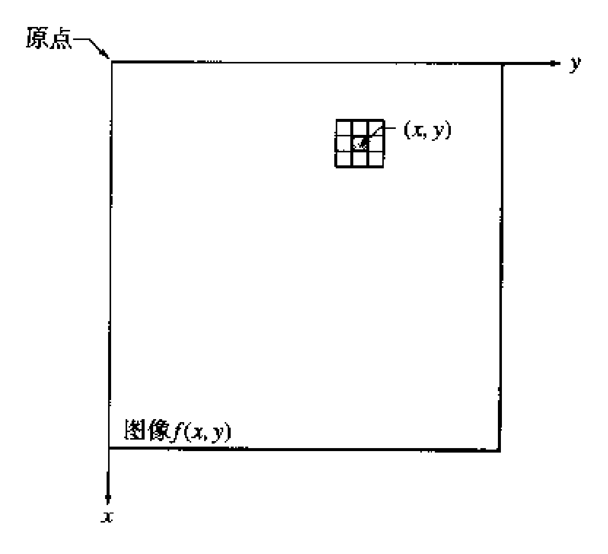


图3.1　图像中点(x, y)的邻域，其大小为3×3

本章的剩余部分将使用前而这个等式来处理各种实现。尽管该等式在概念上很简单，但在MATLAB中，其计算实现要求我们务必注意数据类和取值范围。

3.2　亮度变换函数

变换T的最简单形式是图3.1中的邻域大小为1 × 1（单个像素）时。此时，(x, y)处的g值仅由f在该点处的亮度决定，T也变为一个亮度或灰度级变换函数。当处理单色（灰度）图像时，这两个术语是可以相互换用的。当处理彩色图像时，亮度用来表示某个色彩空间中的一个彩色图像分量，详见第6章中的解释。

由于亮度变换函数仅取决于亮度的值，而与(x, y)无关，所以亮度变换函数通常可写做如下所示的简单形式：

$$s = T(r)$$

其中，r表示图像f中相应点(x, y)的亮度，s表示图像g中相应点(x, y)的亮度。

3.2.1　函数 imadjust

函数imadjust是对灰度图像进行亮度变换的基本IPT工具。其语法为

```
g = imadjust(f, [low_in high_in], [low_out high_out], gamma)
```

正如图3.2中所示的那样，此函数将图像f中的亮度值映像到g中的新值，即将low_in至high_in之间的值映射到low_out至high_out之间的值。low_in以下与high_in以上的值则被剪切掉了；换言之，low_in以下的值映射为low_out，high_in以上的值映射这high_out。输入图像应为uint8类、uint16类或double类图像，输出图像与输入图像有着相同的类。除图像f外，函数imadjust的所有输入输均指定在0和1之间，而不论图像f的类。若f是uint8类图像，则函数imadjust将乘以255来确定应用中的实际值；若f是uint16类图像，则函数imadjust将乘以65 535。为[low_in high_in]或[low_out high_out]使用空矩阵（[ ]）会得到默认值[0 1]。若high_out小于low_out，则输出亮度会反转。

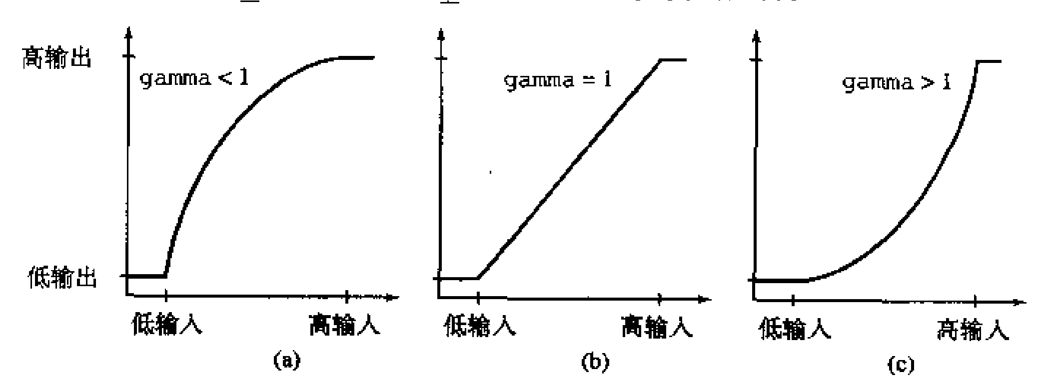


图3.2　函数imadjust中的各种可用映射

参数gamma指定了曲线的形状，该曲线用来映射f的亮度值，以便生成图像g。若gamma小于1，则映射被加权至更高（更亮）的输出值，如图3.2(a)所示。若gamma大于1，则映射被加权至更低（更暗）的输出值。若省略了函数的参量，则gamma默认为1（线性映射）。

例3.1　使用函数imadjust

图3.3(a)是一幅数字乳房X射线图像f，它显示出了一处疾患；图3.3(b)是使用如下命令得到的明暗反转图像（负片图像）：

```
>> g1 = imadjust(f, [0 1], [1 0]);
```

这种获得明暗反转图像的过程可用于增强嵌入在大片黑色区域中的白色或灰色细节。例如，在图3.3(b)中就可非常容易地分析乳房组织。一幅图像的负片同样也可使用IPT函数imcomplement来得到：

```
g = imcomplement(f)
```

图 3.3(c)是执行如下命令后的结果：

```
>> g2 = imadjust(f, [0.5 0.75], [0 1]);
```

该命令将0.5至0.75之间的灰度级扩展到范围[0, 1]。这种类型的处理过程可用于突出我们感兴趣的亮度带。最后，使用命令

```
>> g3 = imadjust(f, [ ], [ ], 2);
```

可获得类似于图3.3(c)所示的结果（但有着更多的灰色调），方法是压缩灰度级的低端并扩展灰度级的高端[见图 3.3(d)]。

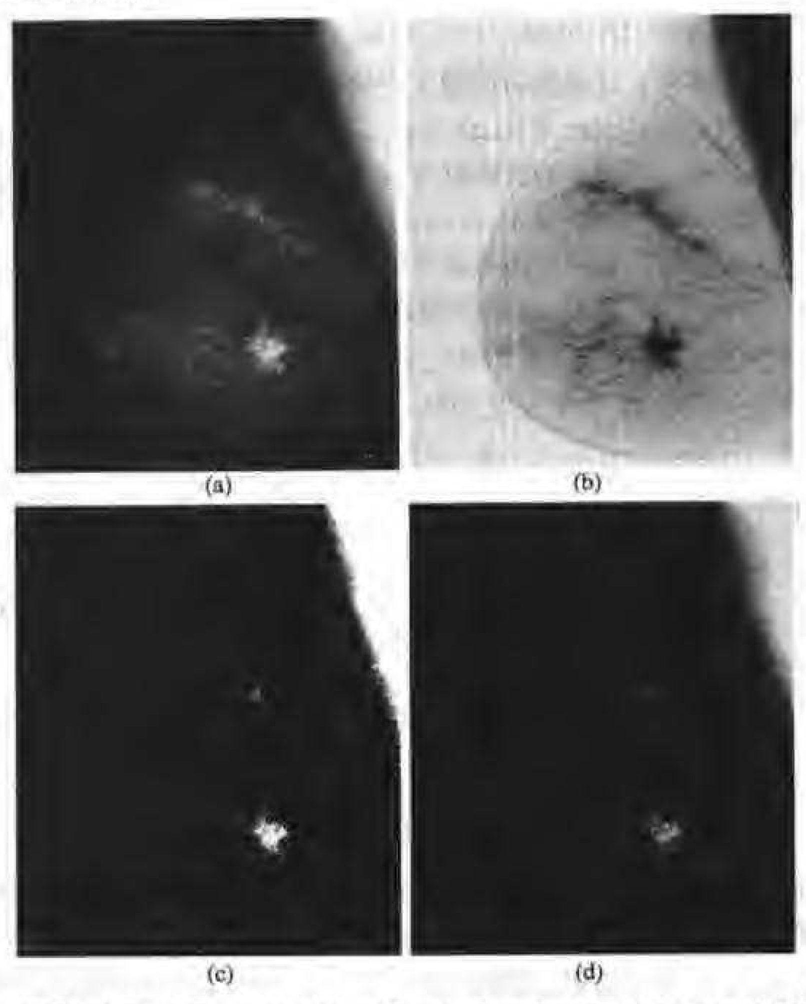


图 3.3 (a)原始数字乳房图像；(b)负片图像；(c)亮度范围扩展为[0.5, 0.75]后的结果；(d)使用 gamma = 2 增强图像后的结果（原图像由 G.E.医学系统公司提供）

3.2.2 对数和对比度拉伸变换

对数与对比度拉伸变换是进行动态范围处理的基本工具。对数变换通过如下表达式实现：

```
g = c*log(1 + double(f))
```

其中，c 是一个常数。该变换的形状类似于图 3.2(a)所示的 gamma 曲线，在两个坐标轴上，低值设为 0，高值设为 1。注意，gamma 曲线的形状是可变的，而对数函数形状则是固定的。

对数变换的一项主要应用是压缩动态范围。例如，傅里叶频谱（见第 4 章）的范围为$[0, 10^6]$或更高。当傅里叶频谱显示于已线性缩放至 8 比特的监视器上时，高值部分占优，从而导致频谱中低亮度值的可视细节丢失。通过计算对数，10^6左右的动态范围会降至 14，从而就更便于我们的处理。

当执行一个对数变换时，我们通常期望将导致的压缩值还原为显示的全范围。对 8 比特而言，在 MATLAB 中这样做的最简方法是使用语句

```
>> gs = im2uint8(mat2gray(g));
```

使用函数 mat2gray 可将值限定在范围[0, 1]内，使用函数 im2uint8 可将值限定在范围[0, 255]内。在 3.2.3 节中，我们将讨论一种比例换算函数，该函数可以自动检测出输入数据的类别并采用相应的转换。

图 3.4(a)所示的函数称为对比度拉伸变换函数，因为该函数可将输入值低于m的灰度级压缩为输出图像中较暗灰度级的较窄范围内；类似地，该函数可将输入值高于m的灰度级压缩为输出图像中较亮灰度级的较窄范围内。输出的是一幅具有高对比度的图像。实际上，在图 3.4(b)所示的限制条件下，输出是一幅二值图像。这种限制函数称为阈值函数（我们将在第 10 章中讨论），它是进行图像分割的一种有效工具。使用本节开头介绍的表示法，图 3.4(a)所示函数的形式为

$$s = T(r) = \frac{1}{1 + (m/r)^E}$$

其中，r表示输入图像的亮度，s是输出图像中的相应亮度值，E控制该函数的斜率。在 MATLAB 中，该式由如下语句对整幅图像完成操作：

```
g = 1./(1 + (m./(double(f) + eps)).^E)
```

注意，使用 eps（见表 2.10）可避免 f 出现 0 值时的溢出现象。由于$T(r)$的限制值为 1，所以在执行此类变换时，输出值也被缩放在范围[0, 1]内。图 3.4(a)所示的曲线形状是使用 E = 20 时获得的。

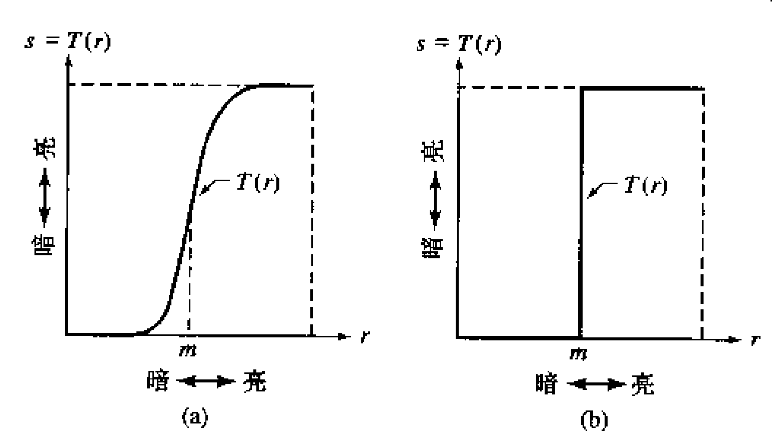


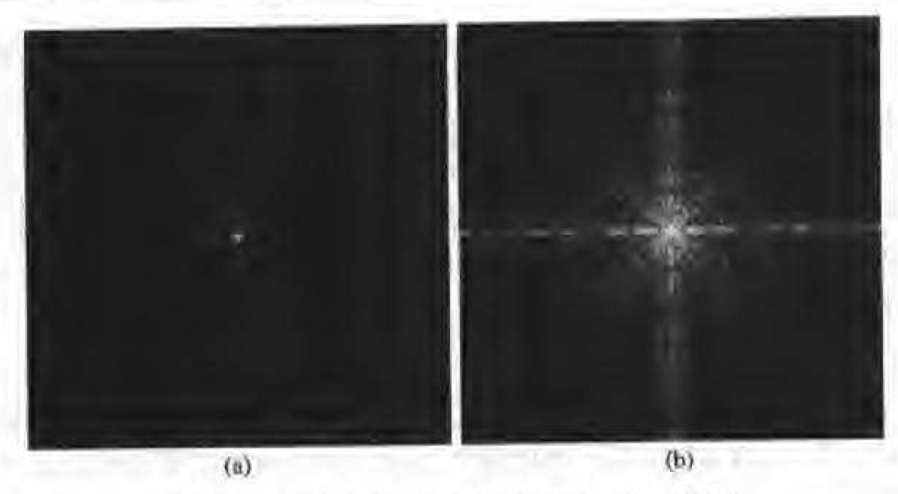
图 3.4　(a)对比度拉伸变换；(b)阈值变换

例 3.2　使用对数变换减小动态范围

图 3.5(a)是一个取值范围为 0 至1.5×10^6的傅里叶频谱，它显示在线性比例尺的 8 比特系统上。图 3.5(b)显示了使用命令

```
>> g = im2uint8(mat2gray(log(1 + double(f))));
>> imshow(g)
```

获得的结果。图像g与原图像相比，在视觉方面的改善效果是非常明显的。



(a) (b)

图3.5 (a)傅里叶频谱；(b)执行对数变换后的结果

3.2.3 亮度变换的一些实用M函数

在本节中，我们将开发两个M函数，它们包含了前两节中介绍的亮度变换的许多内容。我们将给出其中一个函数的具体代码来示例错误检验，介绍几种MATLAB函数的公式化表示方法，以便处理可变数目的输入和/或输出，并说明贯穿全书的典型代码格式。从现在起，仅当我们需要解释特定程序的结构时，才对新M函数的具体代码加以讨论，以便说明新MATLAB或IPT函数的功能，或复习前面介绍过的概念。在其他情况下，我们将只解释函数的语法，而其代码包含在附录C中。此外，为了重点讨论本书剩余部分中已开发函数的基本结构，本节将是说明错误检验广泛用途的最后一节。接下来的步骤是在MATLAB中如何对错误处理进行编程。

处理可变数量的输入和/或输出

为检测输入到M函数的参量数目，我们可使用函数nargin，

```
n = nargin
```

该函数将返回输入到M函数的参量的实际数目。类似地，函数nargout用于M函数的输出。其语法为

```
n = nargout
```

例如，假设我们在提示符处执行如下M函数：

```
>> T = testhv(4, 5);
```

该函数体中使用nargin返回2，使用nargout返回1。

函数nargchk可用于一个M函数体中，以检测传递的参量数目是否正确。其语法为

```
msg = nargchk(low, high, number)
```

该函数在number小于low时，返回消息Not enough input parameters；在number大于high时，返回消息Too many input parameters。若number介于low与high之间，则函数nargchk返回一个空矩阵。若输入参量的数目不正确，则频繁使用函数nargchk可通过函数error来终止程序的执行。实际输入参量的数目由函数nargin决定。例如，考虑下列代码片断：

在下面名为 intrans 的 M 函数中，要注意函数的选项是如何在代码的帮助部分中格式化的，输入的变量数是如何处理的，错误检验是如何插入代码的，以及输入图像的类是如何与输出图像的类相匹配的。记住，在研究如下的代码时，varargin 是一个单元数组，故其元素应使用大括号括起。

```
function g = intrans(f, varargin)
%INTRANS Performs intensity (gray-level) transformations.
%   G = INTRANS(F, 'neg') computes the negative of input image F.
%
%   G = INTRANS(F, 'log', C, CLASS) computes C*log(1 + F) and
%   multiplies the result by (positive) constant C. If the last two
%   parameters are omitted, C defaults to 1. Because the log is used
%   frequently to display Fourier spectra, parameter CLASS offers the
%   option to specify the class of the output as 'uint8' or
%   'uint16'. If parameter CLASS is omitted, the output is of the
%   same class as the input.
%
%   G = INTRANS(F, 'gamma', GAM) performs a gamma transformation on
%   the input image using parameter GAM (a required input).
%
%   G = INTRANS(F, 'stretch', M, E) computes a contrast-stretching
%   transformation using the expression 1./(1 + (M./(F +
%   eps)).^E). Parameter M must be in the range [0, 1]. The default
%   value for M is mean2(im2double(F)), and the default value for E
%   is 4.
%
%   For the 'neg', 'gamma', and 'stretch' transformations, double
%   input images whose maximum value is greater than 1 are scaled
%   first using MAT2GRAY. Other images are converted to double first
%   using IM2DOUBLE. For the 'log' transformation, double images are
%   transformed without being scaled; other images are converted to
%   double first using IM2DOUBLE.
%
%   The output is of the same class as the input, except if a
%   different class is specified for the 'log' option.

% Verify the correct number of inputs.
error(nargchk(2, 4, nargin))
% Store the class of the input for use later.
classin = class(f);
% If the input is of class double, and it is outside the range
% [0, 1], and the specified transformation is not 'log', convert the
% input to the range [0, 1].
if strcmp(class(f), 'double') & max(f(:)) > 1 & . . .
      ~strcmp(varargin{1}, 'log')
   f = mat2gray(f);
else % Convert to double, regardless of class(f).
   f = im2double(f);
end

% Determine the type of transformation specified.
```

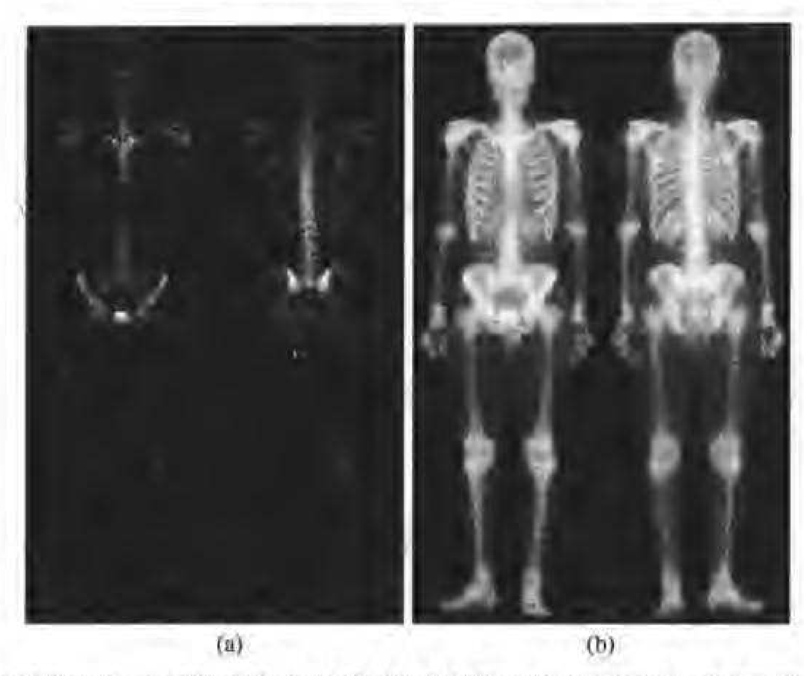
```
method = varargin{1};
% Perform the intensity transformation specified.
switch method
case 'neg'
   g = imcomplement(f);
case 'log'
   if length(varargin) == 1
      c = 1;
   elseif length(varargin) == 2
      c = varargin{2};
   elseif length(varargin) == 3
      c = varargin{2};
      classin = varargin{3};
   else
      error('Incorrect number of inputs for the log option.')
   end
   g = c*(log(1 + double(f)));
case 'gamma'
   if length(varargin) < 2
      error('Not enough inputs for the gamma option.')
   end
   gam = varargin{2};
   g = imadjust(f, [ ], [ ], gam);
case 'stretch'
   if length(varargin) == 1
      % Use defaults.
      m = mean2(f);
      E = 4.0;
   elseif length(varargin) == 3
      m = varargin{2};
      E = varargin{3};
   else error('Incorrect number of inputs for the stretch option.')
   end
   g = 1./(1 + (m./(f + eps)).^E);
otherwise
   error('Unknown enhancement method.')
end
% Convert to the class of the input image.
g = changeclass(classin, g);
```

例3.3 函数 intrans 的说明

要说明函数intrans，我们可首先考虑图3.6(a)所示的图像，这是一幅利用对比度拉伸方法增强骨骼结构后的理想图像。利用下列对函数intrans的调用得到了增强后的图像，如图3.6(b)所示：

```
>> g = intrans(f, 'stretch', mean2(im2double(f)), 0.9);
>> figure, imshow(g)
```

注意函数mean2是如何直接在函数调用内计算f的平均值的。其产生的值为m所用。为了将其值标度在范围[0, 1]内，我们使用函数im2double将图像f转换成了double类图像，从而使平均值m也在此范围内。E的值可交互式地确定。



(a) (b)

图 3.6 (a)骨骼扫描图像；(b)使用对比度拉伸变换增强的图像（原图像由 G.E.医学系统公司提供）

亮度标度的 M 函数

当处理图像时，像素值域由负到正的现象是很普通的。尽管在中间计算过程中这没有问题，但当我们想利用 8 比特或 16 比特格式保存或查看一幅图像时，就会出现问题。在这种情况下，我们通常希望把图像标度在全尺度，即最大范围[0, 255]或[0, 65 535]。下列名为 gscale 的 M 函数能实现此项功能。此外，此函数能将输出映射到一个特定的范围。该函数的代码没无新意，因而在此不再给出。代码的详细清单请参阅附录 C。

函数 gscale 的语法为

```
g = gscale(f, method, low, high)
```

其中，f 是将被标度的图像，method 的有效值为 'full8'（默认）和 'full16'，前者将输出标度为全范围[0, 255]，后者将输出标度为全范围[0, 65 535]。若使用这两个有效值之一，则可在两种变换中省略参数 low 与 high。method 的第三个有效值为 'minmax'，此时，我们必须给出 low 与 high 在范围[0, 1]内的值。若选用的是 'minmax'，则映射的结果值须在范围[low, high]内。尽管这些值均限定在范围[0, 1]内，但程序本身会根据输入的类来执行适当的标度，然后将输出转换为与输入相同的类。例如，若 f 为 uint8 类图像，且限定 'minmax' 在范围[0, 0.5]内，则输出图像同样为 uint8 类图像，其值在范围[0, 128]内。若 f 为 double 类图像，且其值在范围[0, 1]外，则程序在运行之前会将其转换到范围[0, 1]内。本书中的许多地方都用到了函数 gscale。

3.3 直方图处理与函数绘图

基于从图像亮度直方图中提取的信息的亮度变换函数，在诸如增强、压缩、分割、描述等方面的图像处理中扮演着基础性的角色。本节的重点在于获取、绘图并利用直方图技术进行图像增强。直方图的其他应用将在后续章节中加以介绍。

3.3.1 生成并绘制图像的直方图

一幅数字图像在范围[0, G]内总共有 L 个灰度级，其直方图定义为离散函数

$$h(r_k) = n_k$$

其中，r_k是区间[0, G]内的第k级亮度，n_k是灰度级为r_k的图像中的像素数。对于uint8类图像，G的值为255；对于uint16类图像，G的值为65 535；对于double类图像，G的值为1.0。记住，MATLAB中的索引不能为0，故r_1相当于灰度级0，r_2相当于灰度级1，如此等等，r_L相当于灰度级G。其中，uint8类图像或uint16类图像中$G = L - 1$。

通常，我们会用到归一化直方图，即使用所有元素$h(r_k)$除以图像中的像素总数n所得到的图形：

$$p(r_k) = \frac{h(r_k)}{n} = \frac{n_k}{n}$$

其中，$k = 1, 2, \cdots, L$。由基本的概率论知识，我们可知$p(r_k)$表示灰度级r_k出现的频数。

在处理图像直方图的工具箱中，核心函数是imhist，其基本语法为

```
h = imhist(f, b)
```

其中，f为输入图像，h为其直方图$h(r_k)$，b是用于形成直方图的"收集箱"的个数（即灰度级的个数——译者注）。若b未包含在此变量中，则其默认值为256。一个"收集箱"仅仅是亮度标度范围的一小部分。例如，若我们要处理一幅uint8类图像并令b = 2，则亮度标度范围被分成两部分：0至127和128至255。所得的直方图将有两个值：$h(1)$等于图像在区间[0, 127]内的像素总数，$h(2)$等于图像在区间[128，255]内的像素总数。使用表达式

```
p = imhist(f, b)/numel(f)
```

就可简单地获得归一化直方图。函数numel(f)给出数组f中的元素个数（即图像中的像素数），详见2.10.3节的讨论。

例 3.4　计算并绘制图像直方图

考虑图3.3(a)所示的图像f。绘制其直方图的最简方法是使用未指定输出的函数imhist：

```
>> imhist(f) ;
```

图3.7(a)显示了结果。这是工具箱中默认显示的直方图。但绘制直方图的方法还有许多，在此我们将解释MATLAB中的一些有代表性的绘制选项。

直方图经常利用条形图来绘制。为这到此目的，我们使用函数

```
bar(horz, v, width)
```

其中，v是一个行向量，它包含将被绘制的点；horz是一个与v有着相同维数的向量，它包含水平标度值的增量；width是一个值在0和1之间的数。若省略horz，则水平轴会从0至length(v)等分为若干个单位。当width的值为1时，竖条较明显；当width的值为0时，竖条是简单的垂直线，如图3.7(a)所示。width的默认值为0.8。绘制条形图时，我们通常会将水平轴等分为几段，以便降低水平轴的分辨率。下面的语句将生成一幅条形图，其水平轴以10个灰度级为一组：

```
>> h = imhist(f);
>> h1 = h(1:10:256);
>> horz = 1:10:256;
>> bar(horz, h1)
```

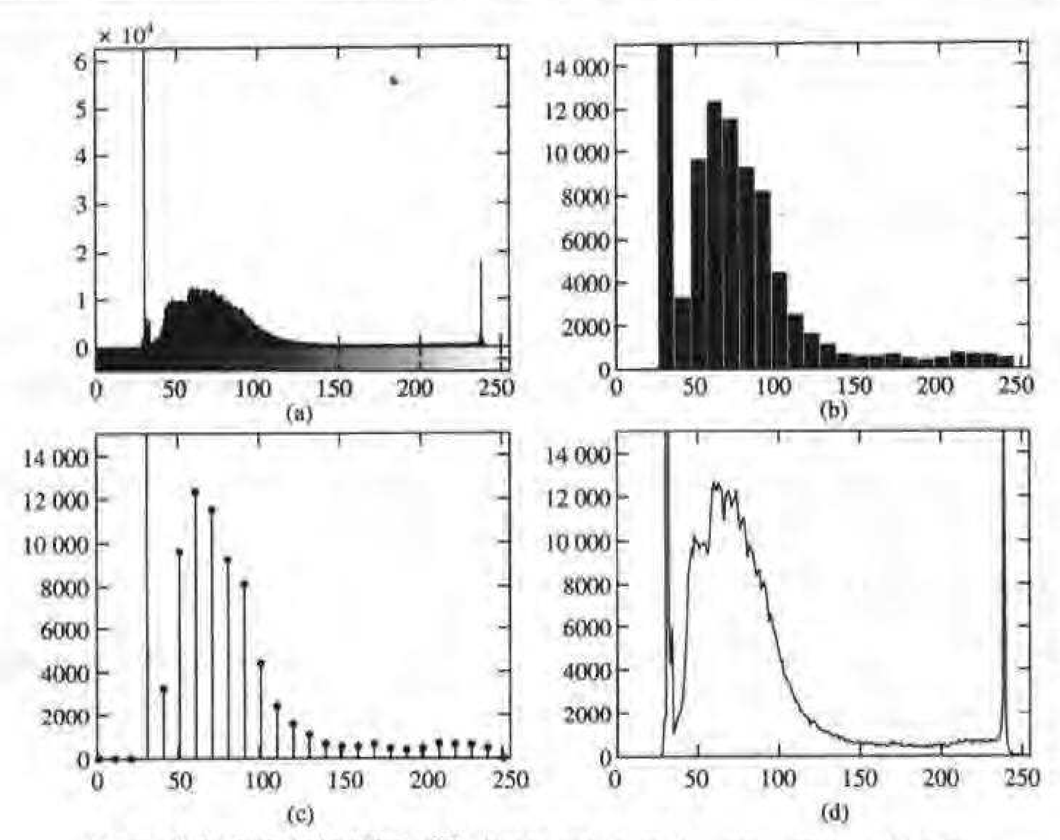


图3.7 绘制图像直方图的各种方法：(a)imhist；(b)bar；(c)stem；(d)plot

```
>> h = imhist(f);
>> h1 = h(1:10:256);
>> horz = 1:10:256;
>> stem(horz, h1, 'fill')
>> axis([0 255 0 15000])
>> set(gca, 'xtick', [0:50:255])
>> set(gca, 'ytick', [0:2000:15000])
```

最后，我们考虑plot函数，该函数将一组点用直线连接起来。其语法为

```
plot(horz, v, 'color_linestyle_marker')
```

其中的各变量在先前介绍杆状图时已定义。color，linestyle和marker的取值在表3.1中给出。如同函数stem那样，函数plot的属性也由三个值指定。当为linestyle或marker选用none时，必须分别指定属性。例如，命令

```
>> plot(horz, v, 'color', 'g', 'linestyle', 'none', 'marker', 's')
```

将绘出中间无连线的绿色方形。plot的默认值为黑色无标记实线。

利用如下语句可得到如图3.7(d)所示的图形：

```
>> h = imhist(f);
>> plot(h)    % Use the default values.
>> axis([0 255 0 15000])
>> set(gca, 'xtick', [0:50:255])
>> set(gca, 'ytick', [0:2000:15000])
```

函数plot经常频繁用于显示变换函数（见例3.5）。

在前面的讨论中，坐标轴的取值范围和刻度线都是人工设定的。使用函数 ylim 和函数 xlim 可以自动设定坐标轴的取值范围和刻度线。此时我们有语法形式

```
ylim('auto')
xlim('auto')
```

这两个函数（详见在线帮助）也有人工设定选项，即

```
ylim([ymin  ymax])
xlim([xmin  xmax])
```

它允许我们手工设置取值范围。若只对一个轴设置取值范围，则另一个轴将默认设置为 'auto'。我们将在下一节中使用这些函数。

在提示符处键入 hold on 将保留当前的图形及某些轴属性，以便使后续绘图命令可在已有图形上执行。详见例 10.6 中的说明。

3.3.2 直方图均衡化

假设灰度级为归一化至范围[0, 1]内的连续量，并令 $p_r(r)$ 表示某给给定图像中的灰度级的概率密度函数（PDF），其下标用来区分输入图像和输出图像的 PDF。假设我们对输入灰度级执行如下变换，得到（处理后的）输出灰度级 s：

$$s = T(r) = \int_0^r p_r(w)\,\mathrm{d}w$$

式中 w 是积分的哑变量。可以看出（Gonzalez and Woods[2002]），输出灰度级的概率密度函数是均匀的，即

$$p_s(s) = \begin{cases} 1, & 0 \leqslant s \leqslant 1 \\ 0, & \text{其他} \end{cases}$$

换言之，前述变换生成一幅图像，该图像的灰度级较为均衡化，且覆盖了整个范围[0, 1]。灰度级均衡化处理的最终结果是一幅扩展了动态范围的图像，它具有较高的对比度。注意，该变换函数只不过是一个累积分布函数（CDF）。

使用直方图并调用直方图均衡化技术来处理离散灰度级时，一般来说，处理后的图像的直方图将不再均匀，这源于变量的离散属性。参考在 3.3.1 节中的讨论，令 $p_r(r_j), j = 1, 2, \cdots, L$ 表示与给定图像的灰度级相关的直方图，并回忆归一化直方图中的各值大致是图像取各灰度级的概率。对于离散的灰度级，我们采用求和方式，且均衡化变换变为

$$\begin{aligned} s_k &= T(r_k) \\ &= \sum_{j=1}^{k} p_r(r_j) \\ &= \sum_{j=1}^{k} \frac{n_j}{n} \end{aligned}$$

式中 $k = 1, 2, \cdots, L$，且 s_k 是输出（处理后的）图像中的亮度值，它对应于输入图像中的亮度值 r_k。

直方图均衡化由工具箱中的函数 histeq 实现，该函数的语法为

```
g = histeq(f, nlev)
```

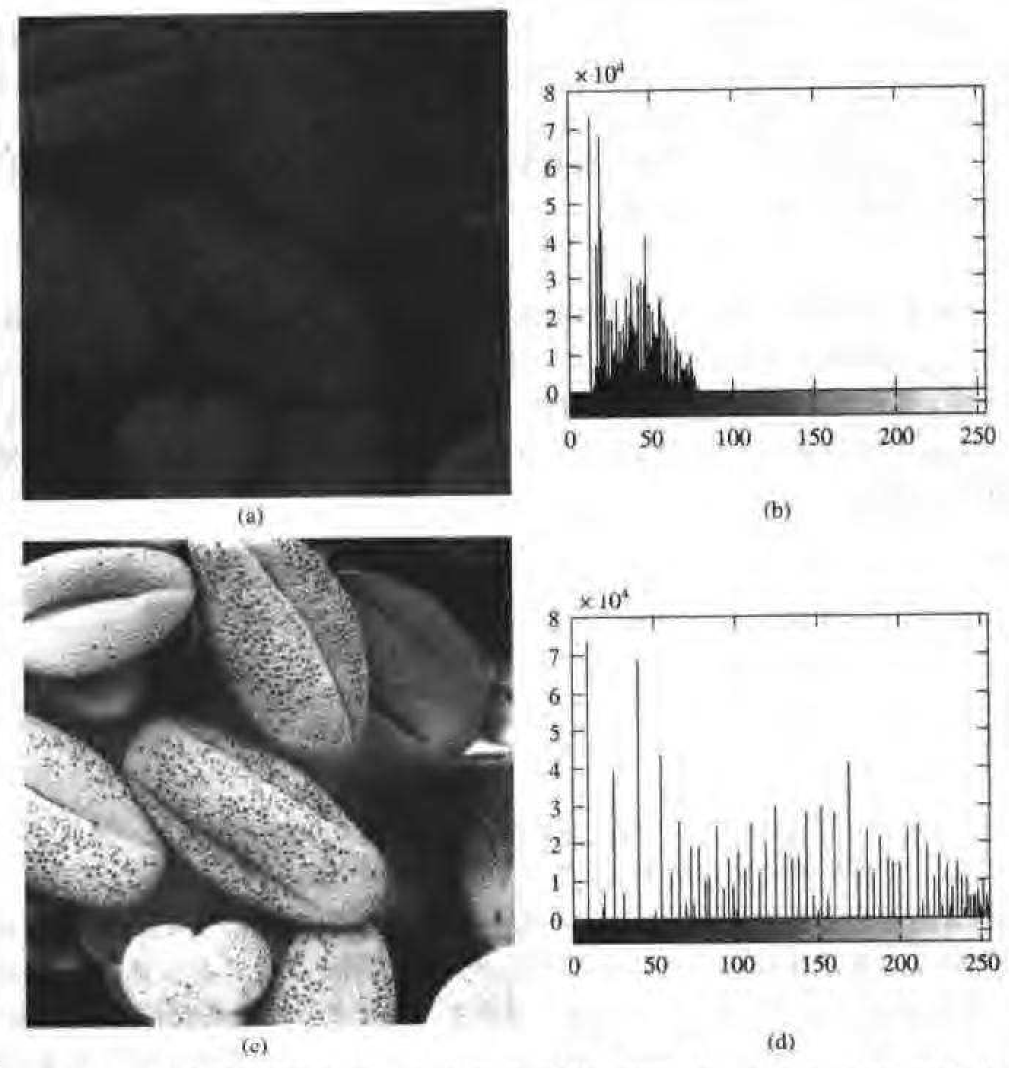


图 3.8　直方图均衡化实例。(a)输入图像；(b)输入图像的直方图；(c)直方图均衡化后的图像；(d)图像均衡化后的直方图。与(a)相比，(c)的增强十分明显（原图像由澳大利亚堪培拉大学生物科学研究院的 Roger Heady 博士提供）

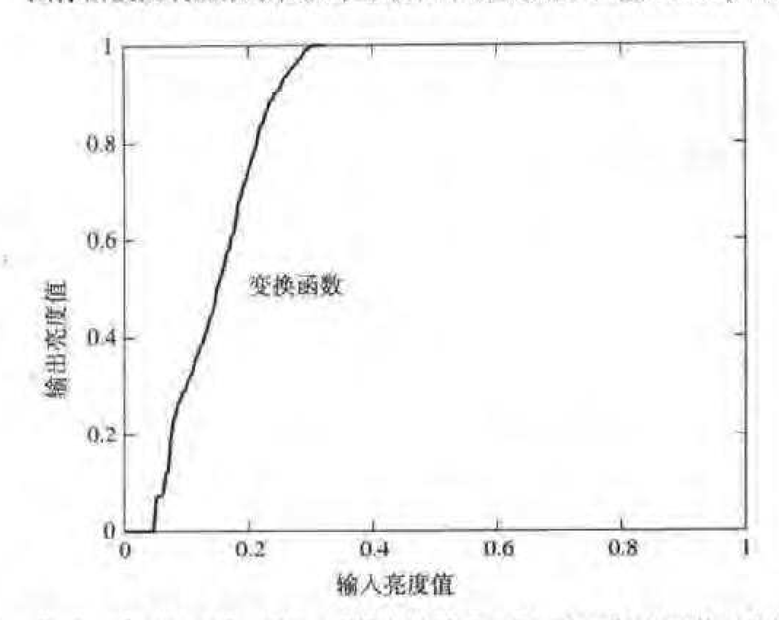


图 3.9　用于将图 3.8(a)所示输入图像映射到图 3.8(c)所示输出图像的变换函数

3.3.3 直方图匹配（规定化）

直方图均衡化生成了自适应的变换函数，从这方面讲，它是以已知图像的直方图为基础的。然而，一旦一幅图像的变换函数已经计算完毕，它将不再改动，除非直方图有变动。正如前面几节中提到的那样，直方图均衡化通过扩展输入图像的灰度级到较宽亮度尺度的范围来实现图像增强。在本节中，我们将说明这种方法有时并不总能得到成功的结果。要特别指出的是，能够指定想要的处理后的图像的直方图形状在某些应用中是非常有用的。生成具有指定直方图的图像的方法称为直方图匹配或直方图规定化。

这种方法的原理很简单。考虑归一化到区间[0, 1]后的连续灰度级，令r和z分别表示输入图像与输出图像的灰度级。输入灰度级的概率密度函数为$p_r(r)$，输出灰度级的概率密度函数为$p_z(z)$。由前几节的讨论我们可知变换

$$s = T(r) = \int_0^r p_r(w)\,\mathrm{d}w$$

导致了有着均衡化概率密度函数$p_s(s)$的灰度级s。假设我们现在定义一个变量z，该变量具有属性

$$H(z) = \int_0^z p_z(w)\,\mathrm{d}w = s$$

记住，我们要得到的是灰度级为z的输出图像，且具有指定的概率密度函数$p_z(z)$。由前面的两个等式可得

$$z = H^{-1}(s) = H^{-1}[T(r)]$$

因为可以由输入图像得到$T(r)$（即上一节中讨论的直方图均衡化变换），所以只要找到H^{-1}，就能使用前面的等式得到变换后的灰度级z，其PDF为指定的$p_z(z)$。当处理离散变量时，若我们能够保证$p_z(z)$是有效的直方图概率密度函数（即该直方图具有单位面积且其所有值非负），则H的逆变换存在，且其元素值非零（即$p_z(z)$中的bin非空）。如同在直方图均衡化中一样，前述方法的离散实现仅产生了指定直方图的近似。

工具箱使用函数`histeq`的如下形式实现直方图匹配：

```
g = histeq(f, hspec)
```

其中，`f`为输入图像，`hspec`为指定的直方图（一个由指定值构成的行向量），`g`为输出图像，其直方图近似于指定的直方图`hspec`。向量中包含对应于等分空间bin的整数值。`histeq`的一个特性是在`length(hspec)`远小于图像`f`中的灰度级数时，图像`g`的直方图通常会较好地匹配`hspec`。

例3.6　直方图匹配

图3.10(a)显示了火星卫星Phobos的一幅图像`f`，图3.10(b)显示了使用`imhist(f)`函数得到的直方图。由于这幅图像中存在大片的较暗区域，所以直方图中的大部分像素都集中在灰度级的暗端。乍一看，人们会认为利用直方图均衡化来增强该图像是一种较好的方式，以便使较暗区域中的细节更加明显。然而，使用命令

```
>> f1 = histeq(f, 256);
```

得到的如图3.10(c)所示的结果表明，利用直方图均衡化方法在本例中并没有得到特别好的结果。对此，通过研究均衡化后的图像的如图3.10(d)所示的直方图可以看出其原因。这里，我们看到灰度级已移动到了灰度级的上半部分，因而输出图像出现了褪色现象。灰度级移动的原因是原始直方图中的暗色分量过于集中在0附近。从而，由该直方图得到的累积变换函数非常陡，因此才把在灰度级低端过于集中的像素映射到了灰度级的高端。

```
while repeats
   s = input('Enter m1, sig1, m2, sig2, A1, A2, k OR x to quit:', 's');
   if s == quitnow
      break
   end
   % Convert the input string to a vector of numerical values and
   % verify the number of inputs.
   v = str2num(s);
   if numel(v) ~= 7
      disp('Incorrect number of inputs.')
      continue
   end
   p = twomodegauss(v(1), v(2), v(3), v(4), v(5), v(6), v(7));
   % Start a new figure and scale the axes. Specifying only xlim
   % leaves ylim on auto.
   figure, plot(p)
   xlim([0 255])
end
```

(a) (b)

(c) (d)

图3.10 (a)火星卫星Phobos的图像；(b)图(a)的直方图；(c)直方图均衡化后的图像；(d)图(c)的直方图（原图像由NASA提供）

由于直方图均衡化在本例中出现的问题主要是原图像0级灰度附近像素过于集中，因而较为合理的手段是修改该图像的直方图，使其不再有此性质。图3.11(a)显示了一个函数的图形（利用程序manualhist得到），它不仅保留了原始直方图的大体形状，而且在图像的较暗区域中灰度级有较为平滑的过渡。程序的输出p由该函数产生的256个等间隔点组成，它是我们所希望的指定直方图。一幅具有指定直方图的图像可由命令

```
>> g = histeq(f, p);
```

生成。图3.11(b)显示了最终结果。比较图3.11(b)与图3.10(c)可知，改进直方图后的结果增强是非常明显的。我们注意到，指定直方图是对原始直方图的恰当更改，而这正是在图像增强方面获得重大改进所要求的。图3.11(b)的直方图如图3.11(c)所示。该直方图最明显的特性是其低端移动到了接近灰度级的较亮区域，从而接近于所指定的形状。但请注意，这里的向右移动幅度远没有图3.10(d)所示直方图的向右移动幅度大，因而得到的图3.10(c)只是一幅增强效果很差的图像。

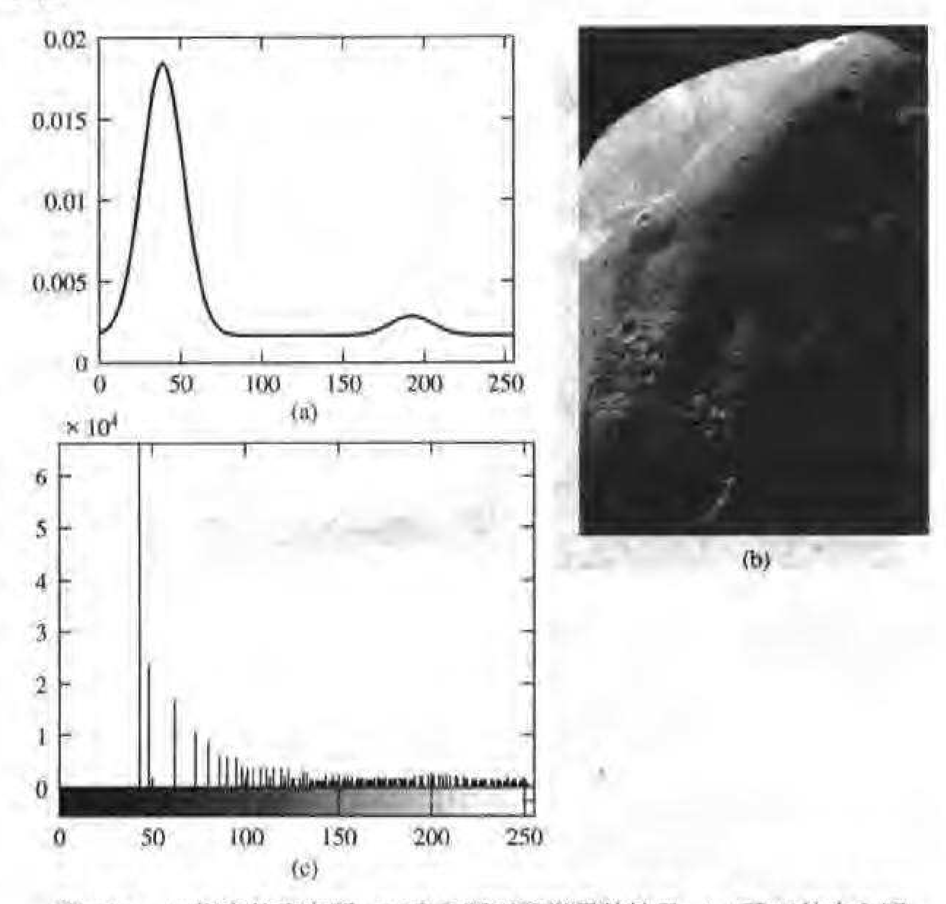


图3.11 (a)指定的直方图；(b)直方图匹配增强的结果；(c) 图(b)的直方图

3.4 空间滤波

正如在3.1节中提到并在图3.1中举例说明的那样，邻域处理包括：(1)定义中心点(x, y)；(2)仅对预先定义的以(x, y)为中心点的邻域内的像素进行运算；(3)令运算结果为该点处处理的响应；(4)对图像中的每一点重复此步骤。移动中心点会产生新的邻域，而每个邻域对应于输入图像上的一个像素。用来区别此过程的两种主要术语为邻域处理和空间滤波，其中第二个术语应用更为普遍。就像

下一节中将要介绍的那样，若对邻域中像素的计算为线性运算时，则此运算称为线性空间滤波（也称为空间卷积）；否则，我们称此运算为非线性空间滤波。

3.4.1　线性空间滤波

线性滤波的概念源于频域中信号处理所使用的傅里叶变换，这一主题将在第4章中详细讨论。在本章中，我们感兴趣的是直接对图像中的像素执行滤波运算。我们使用线性空间滤波这一术语来区分这种类型的处理与频域滤波。

本章所关注的线性运算包括将邻域中每个像素与相应的系数相乘，然后将结果进行累加，从而得到点(x, y)处的响应。若邻域的大小为$m \times n$，则总共需要mn个系数。这些系数排列为一个矩阵，我们称其为滤波器、掩模、滤波掩模、核、模板或窗口，前三种术语最为通用。为简便起见，也常使用卷积滤波、掩模或核等术语。

线性空间滤波的机理如图3.12所示。这个过程仅是简单地在图像f中逐点移动滤波掩模w的中心。在每个点(x, y)处，滤波器在该点处的响应是滤波掩模所限定的相应邻域像素与滤波器系数的乘积结果的累加。对于一个大小为$m \times n$的掩模，假定$m = 2a + 1$且$n = 2b + 1$，其中a和b为非负整数。所有假设都是基于掩模的大小应均为奇数的原则，有意义掩模的最小尺寸是3×3（由于大小为1×1的掩模价值不大，所以在此不予讨论）。尽管并不是一个必须具备的条件，但处理奇数尺寸的掩模会更加直观，因为它们都有惟一的一个中心点。

在执行线性空间滤波时，我们必须清楚理解两个意义相近的概念。一个是相关，另一个是卷积。相关是指掩模w按图3.12所示的方式在图像f中移动的过程。从技术上讲，卷积是相同的过程，只是在图像f中移动w前，要将w旋转180°。这两个概念均可通过几个简单的例子得到解释。

图3.13(a)显示了一个一维函数f和一个掩模w。假设f的原点为其左侧的点。为求两个函数的相关，我们可移动w，使w最右侧的点与f的原点重合，如图3.13(b)所示。注意，这两个函数之间有一些点未重叠。处理这种问题通用方法是在f中填充足够多的零，以保证w在f中移动时，总存在相应的点。这种情形如图3.13(c)所示。

现在我们准备执行相关操作。相关的第一个值是在图3.13(c)所示位置上两个函数乘积的累加和。此时，乘积的累加和为0。接着，我们将w向右移动一个位置并重复上述步骤［见图3.13(d)］。乘积的累加和仍为0。经过4次移动后［见图3.13(e)］，我们首次得到了相关的非零值，即$(2)(1) = 2$。按照这种方式操作下去，直至w全部移过图像f［最终的几何图如图3.13(f)所示］，我们就可得到如图3.13(g)所示的结果。这组值即为w与f的相关。注意，若我固定w而使f在w上移动，则结果将会不同，因而顺序也是有关系的。

在图3.13(g)所示的相关中，符号`'full'`是一个由工具箱使用的标记（稍后将做讨论），表示相关操作按刚描述的方式计算时使用的是经过充零后的图像。工具箱提供了另外一个选项，在图3.13(h)中用`'same'`表示，它产生一个大小与f相同的相关。这种计算同样也使用经过充零后的图像，但起始点位于掩模的中心点（在w中标为3的点），掩模的中心点则与f的原点对齐。最后的计算是使f的最后一个点与掩模的中心点对齐。

要执行卷积，我们可将w旋转180°，并使其最左侧的点与图像f的原点重叠，如图图3.13(j)所示。然后，重复相关计算中所使用的滑动/计算过程，如图3.13(k)到图3.13(n)所示。`'full'`和`'same'`卷积结果分别显示在图3.13(o)和图3.13(p)中。

图3.13中的函数f是一个离散单位冲击函数，该函数在某个位置值为1，而在其他位置值为0。从图3.13(o)或图3.13(p)中的结果可以很明显地看出，卷积基本上只是简单地在冲击的位置复制w。这个简单的复制性质（称为筛选）是线性系统理论中的一个基本概念，也是其中一个函数总会在卷

积中旋转 180° 的原因。注意，与相关不同的是，颠倒该函数的顺序会产生相同的卷积结果。若函数对称移动，则卷积和相关操作会产生相同的结果。

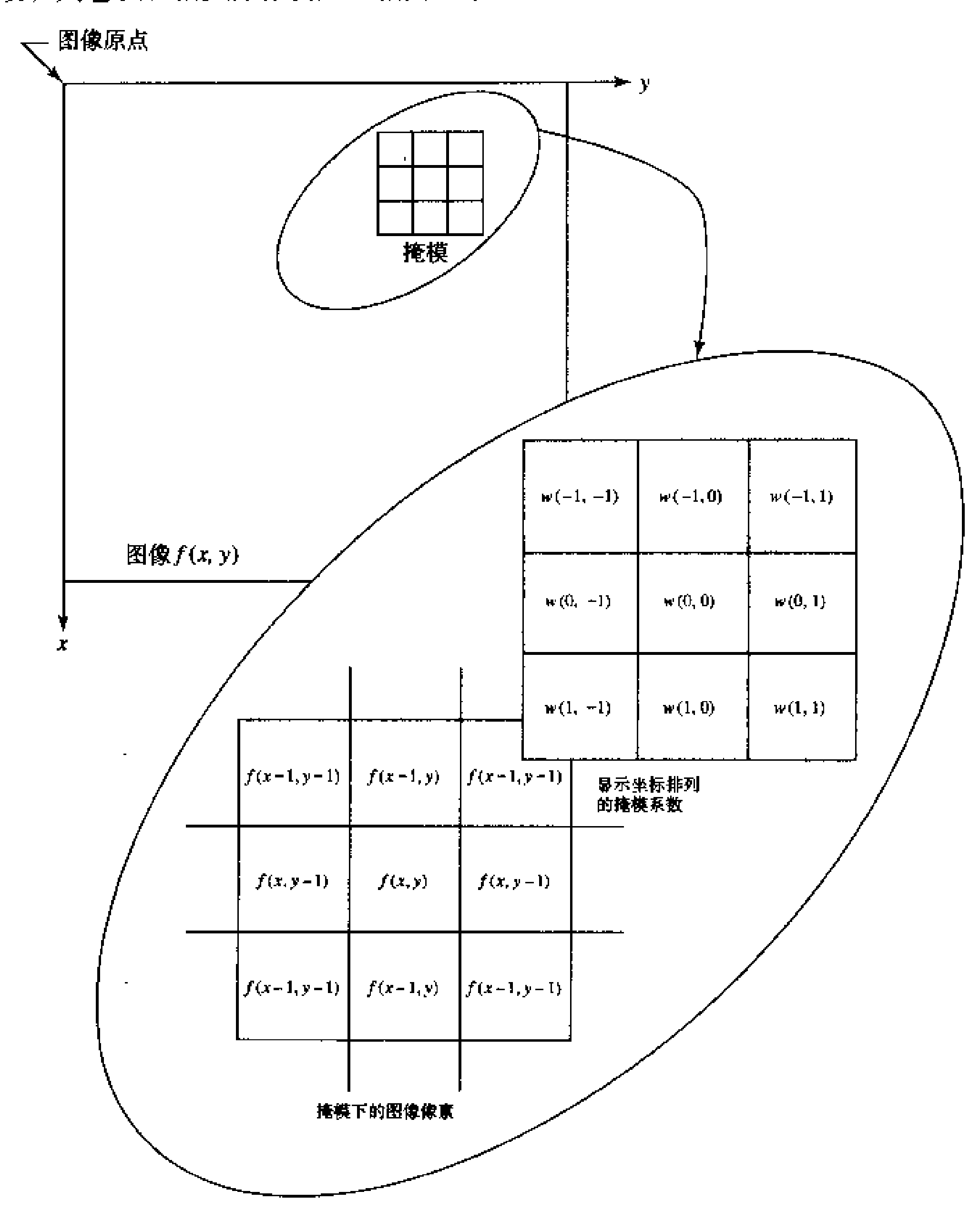


图 3.12　线性空间滤波的机理。放大图显示了大小为 3 × 3 的掩模以及掩模正下方的相应图像邻域。将邻域从掩模正下方移开的目的是便于我们阅读

前面的概念可以很容易推广到图像中，如图 3.14 所示。原点位于位于图像 $f(x, y)$ 的左上角（见图 2.1）。为执行相关计算，我们设置 $w(x, y)$ 的右下角点，以便它与 $f(x, y)$ 的原点重合，如图 3.14(c) 所示。注意，基于在图 3.13 中讨论的原因，我们使用了零填充。为执行相关计算，我们要在所有可能的位置上移动 $w(x, y)$，以便使得它的至少一个像素与原图像 $f(x, y)$ 中的一个像素相重叠。`'full'` 相关示于图 3.14(d) 中。为得到图 3.14(e) 中所示的 `'same'` 相关，我们要求 $w(x, y)$ 的所有偏移都能实现其中心像素覆盖原图像 $f(x, y)$。

对卷积，我们只须简单地将 $w(x, y)$ 旋转 180°，其他处理方式与相关操作相同[见图 3.14(f) 到图 3.14(h)]。正如前面讨论的一维示例那样，若不考虑这两个函数中的哪一个正在进行平移，则卷

积会产生同样的结果。在相关中，顺序也很重要，若假设滤波掩模总是进行平移的函数，则在工具箱中就可以很清楚地看到这一事实。还要注意图3.14(e)和图3.14(h)显示的重要事实，即空间相关和卷积的结果彼此之间旋转了180°。当然，这是我们所期望的结果，因为使用旋转后的滤波掩模时，卷积就是相关。

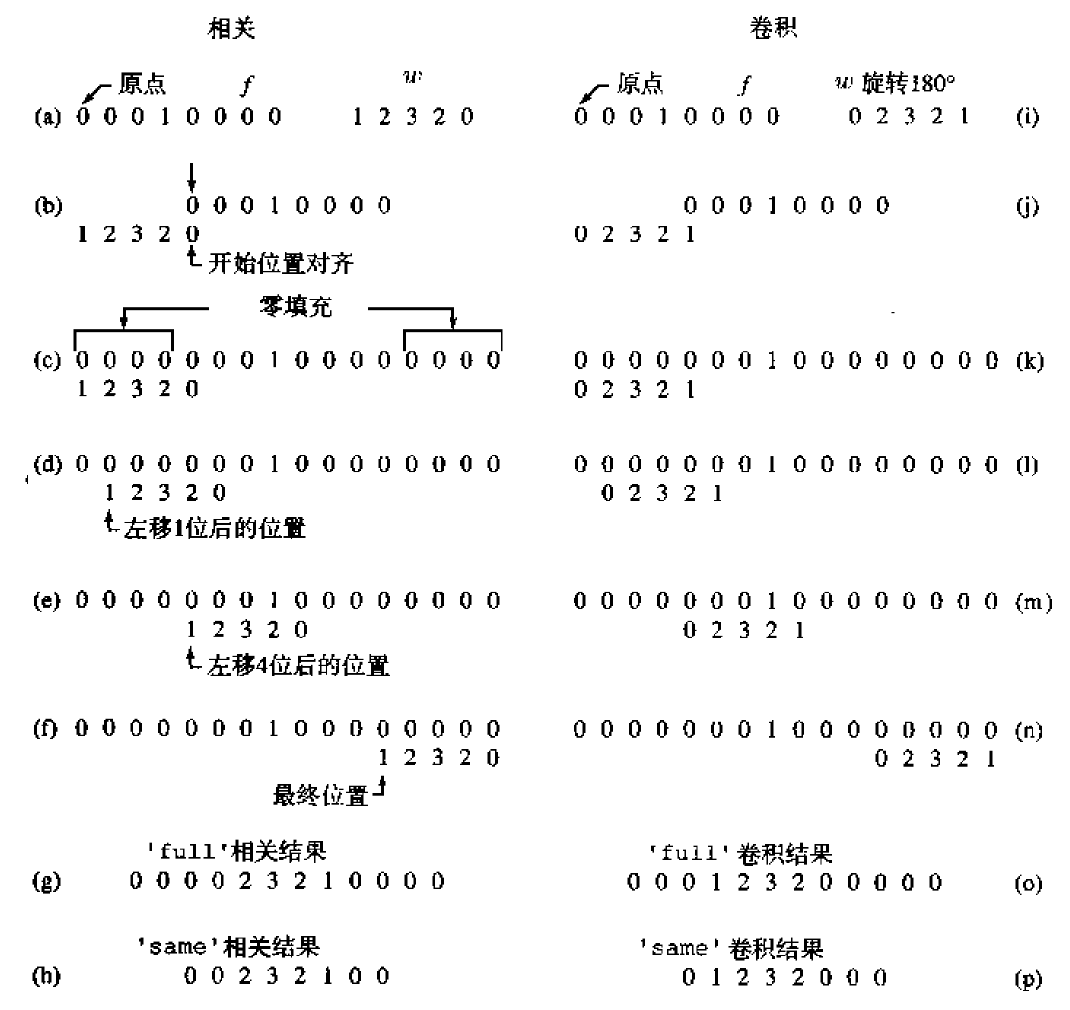


图3.13　一维相关和卷积操作示例

工具箱使用函数 imfilter 来实现线性空间滤波，该函数的语法为

```
g = imfilter(f, w, filtering_mode, boundary_options, size_options)
```

其中，f 是输入图像，w 为滤波掩模，g 为滤波结果，其他参数总结在表3.2中。filtering_mode 用于指定在滤波过程中是使用相关（'corr'）还是卷积（'conv'）。boundary_options 用于处理边界充零问题，边界的大小由滤波器的大小确定。这些选项将在例3.7中详细解释。size_options 可以是 'same' 或 'full'，如图3.13和图3.14所示。

函数 imfilter 的通用语法为

```
g = imfilter(f, w, 'replicate')
```

在实现IPT标准线性空间滤波时，会使用到该语法。这些将在3.5.1节中详细探讨的滤波器要预先旋转180°，以便我们可在 imfilter 中使用默认的相关。由对图3.14的讨论，我们知道对旋转过的滤波器执行相关操作与对原始滤波器进行卷积操作是相同的。若滤波器关于其中心对称，则两个选项将产生同样的结果。

```
                              填充后的 f
                              0 0 0 0 0 0 0 0 0
                              0 0 0 0 0 0 0 0 0
                              0 0 0 0 0 0 0 0 0
  f(x, y) 的原点               0 0 0 0 0 0 0 0 0
  0 0 0 0 0                   0 0 0 0 1 0 0 0 0
  0 0 0 0 0     w(x, y)       0 0 0 0 0 0 0 0 0
  0 0 1 0 0      1 2 3        0 0 0 0 0 0 0 0 0
  0 0 0 0 0      4 5 6        0 0 0 0 0 0 0 0 0
  0 0 0 0 0      7 8 9        0 0 0 0 0 0 0 0 0
        (a)                           (b)

  w 的初始位置                 'full' 相关结果             'same' 相关结果
  1 2 3 0 0 0 0 0 0           0 0 0 0 0 0 0 0 0           0 0 0 0 0
  4 5 6 0 0 0 0 0 0           0 0 0 0 0 0 0 0 0           0 9 8 7 0
  7 8 9 0 0 0 0 0 0           0 0 0 0 0 0 0 0 0           0 6 5 4 0
  0 0 0 0 0 0 0 0 0           0 0 0 9 8 7 0 0 0           0 3 2 1 0
  0 0 0 0 1 0 0 0 0           0 0 0 6 5 4 0 0 0           0 0 0 0 0
  0 0 0 0 0 0 0 0 0           0 0 0 3 2 1 0 0 0
  0 0 0 0 0 0 0 0 0           0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0           0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0           0 0 0 0 0 0 0 0 0
        (c)                          (d)                         (e)

  旋转后的 w                   'full' 卷积结果             'same' 卷积结果
  9 8 7 0 0 0 0 0 0           0 0 0 0 0 0 0 0 0           0 0 0 0 0
  6 5 4 0 0 0 0 0 0           0 0 0 0 0 0 0 0 0           0 1 2 3 0
  3 2 1 0 0 0 0 0 0           0 0 0 0 0 0 0 0 0           0 4 5 6 0
  0 0 0 0 0 0 0 0 0           0 0 0 1 2 3 0 0 0           0 7 8 9 0
  0 0 0 0 1 0 0 0 0           0 0 0 4 5 6 0 0 0           0 0 0 0 0
  0 0 0 0 0 0 0 0 0           0 0 0 7 8 9 0 0 0
  0 0 0 0 0 0 0 0 0           0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0           0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0           0 0 0 0 0 0 0 0 0
        (f)                          (g)                         (h)
```

图 3.14　二维相关和卷积操作示例。使用灰色 0 的目的在于简化观察

表 3.2　函数 imfilter 的选项

选项	描述
滤波类型	
'corr'	滤波通过使用相关来完成（见图 3.13 和图 3.14）。该值是默认值
'conv'	滤波通过使用卷积来完成（见图 3.13 和图 3.14）
边界选项	
P	输入图像的边界通过用值 P（无引号）来填充来扩展。P 的默认值为 0
'replicate'	图像大小通过复制外边界的值来扩展
'symmetric'	图像大小通过镜像反射其边界来扩展
'circular'	图像大小通过将图像看成是一个二维周期函数的一个周期来扩展
大小选项	
'full'	输出图像的大小与被扩展图像的大小相同（见图 3.13 和图 3.14）
'same'	输出图像的大小与输入图像的大小相同。这可通过将滤波掩模的中心点的偏移限制到原图像中包含的点来实现（见图 3.13 和图 3.14）。该值为默认值

在使用预先旋转的滤波器或对称的滤波器时，我们希望执行卷积计算，因而就有两种方法。方法之一是使用语法

```
g = imfilter(f, w, 'conv', 'replicate')
```

方法之二是使用函数rot90(w,2)将w旋转180°，然后使用imfilter(f,w,'replicate')。当然，这两步可以合并为一条语句。前一个语法的图像g的大小与输入图像的大小相同（即计算中的默认模式为前面提到的'same'）。

滤波后的图像的每个元素使用双精度浮点算术进行计算。然而，imfilter会将输出图像转换为与输入图像相同的类。因此，若f是一个整数数组，则输出中超过整型范围的元素将被截断，且小数部分会四舍五入。若结果要求更高的精度，则f需要在使用函数imfilter之前利用im2double或double转换为double类。

例3.7 使用函数imfilter

图3.15(a)是一幅double类图像f，大小为512 × 512像素。考虑一个大小为31 × 31的简单滤波器

```
>> w = ones(31);
```

该滤波器近似为一个平均滤波器。这里我们未用$(31)^2$去除系数，以便在本例的末尾显示使用函数imfilter来处理一幅uint8类图像的缩放效果。

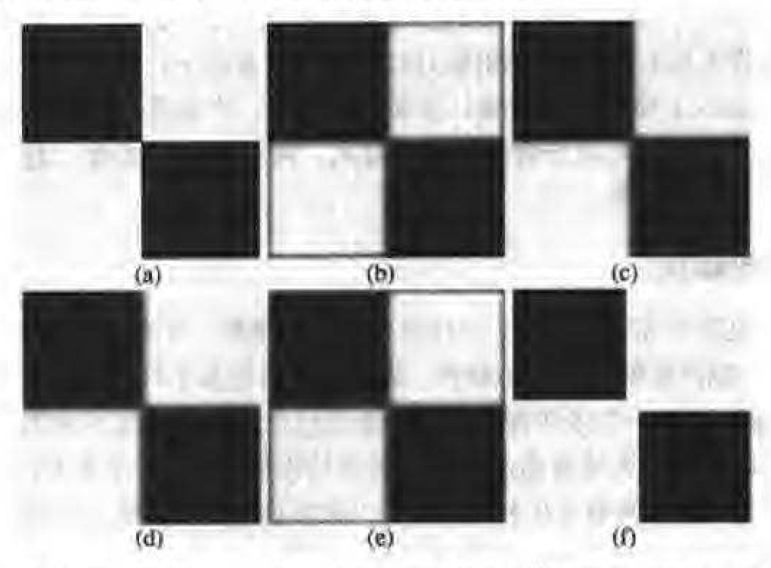


图3.15 (a)原图像；(b)使用默认零填充的imfilter函数后的结果；(c)选项'replicate'的结果；(d)选项'symmetric'的结果；(e)选项'circular'的结果；(f)将原图像转换为uint8类图像，然后使用选项'replicate'进行滤波后的结果。滤波器的大小为31 × 31，其所有元素均为1

用滤波器w对图像进行卷积操作会产生模糊的结果。由于滤波器是对称的，所以我们可以使用imfilter中默认的相关。图3.15(b)显示了执行下述滤波操作后的结果：

```
>> gd = imfilter(f, w);
>> imshow(gd, [ ])
```

其中我们使用了默认的边界选项，该选项使用零（黑色）填充了图像的边界。就像我们预料的那样，滤波后的图像中黑白边缘模糊了，图像中的较亮部分与边界间的边缘也模糊了。当然，原因在于图像用零填充后的边缘为黑色。使用'replicate'选项可以解决这一问题：

```
>> gr = imfilter(f, w, 'replicate');
>> figure, imshow(gr, [ ])
```

如图3.15(c)所示，滤波后的图像的边界正如我们所料想的那样。在这种情况下，使用'symmetric'选项可获得相同的结果：

```
>> gs = imfilter(f, w, 'symmetric');
>> figure, imshow(gs, [ ])
```

图 3.15(d)显示了结果。然而，使用选项 'circular'

```
>> gc = imfilter(f, w, 'circular');
>> figure, imshow(gc, [ ])
```

产生的结果如图3.15(e)所示，其出现的问题与使用零填充的问题相同。这一点与我们所预料的相同，因为周期的使用可使得图像的黑暗部分靠近光亮区域。

最后，我们将说明imfilter产生的结果与输入有着相同类这一事实，会在不当处理时导致问题：

```
>> f8 = im2uint8(f);
>> g8r = imfilter(f8, w, 'replicate');
>> figure, imshow(g8r, [ ])
```

图 3.15(f)显示了上述操作的结果。在这种情况下，当输出通过 imfilter 转换为与输入相同的类（uint8 类）时，剪切会引起数据的丢失。原因是掩模的系数并不在范围[0, 1]内求和，因而会导致滤波后的结果超出范围[0, 255]。为避免这种问题，我们可使用一个用来归一化系数的选项，以便使得系数和限定在范围[0, 1]内（在现有条件下，可以用系数除以$(31)^2$，以便和为 1），或者以 double 格式输入数据。但要注意的是，即使是以 double 格式输入数据，数据仍需要归一化为某个点处的一种有效图像格式。两种方法均有效；关键是要注意数据的范围，以避免出现意外的结果。

3.4.2 非线性空间滤波

非线性空间滤波也基于邻域操作的，且与前一节讨论那样，可通过定义一个大小为$m \times n$的邻域，以其中心点滑过一幅图像的方式进行操作。线性空间滤波基于计算乘积之和（这是一个线性操作），而非线性空间滤波则基于非线性操作，这种操作包含了一个邻域的像素。例如，令每个中心点处的响应等于其邻域内的最大像素值的操作即为非线性滤波。另一个基本区别是，掩模的概念在非线性处理中并不流行。滤波的概念仍然存在，但“滤波器”应看做是一个基于邻域像素操作的非线性函数，其响应组成了在邻域的中心像素处操作的响应。

工具箱提供了两个执行常规非线性滤波的函数，即函数nlfilter和函数colfilt。函数nlfilter直接执行二维操作，而函数colfilt则以列的形式组织数据。虽然colfilt需要占用更多的内存，但是执行起来要比nlfilter快得多。在大多数图像处理应用中，速度是最重要的因素，因此，在执行常规的非线性空间滤波时，我们更多采用的是 colfilt 而不是nlfilt。

给定一个大小为$M \times N$的图像 f 和一个大小为为$m \times n$的邻域，函数colfilt生成一个最大尺寸为$mn \times MN$[①]的矩阵A，在该矩阵中，每一列对应于其中心位于图像内某个位置的邻域所包围的像素。例如，当邻域的中心位于f的最左上侧时，第一列对应于该邻域所包围的像素。所有要求的填充均由 colfilt 透明地完成（使用零填充）。

函数 colfilt 的语法为

```
g = colfilt(f, [m n], 'sliding', @fun, parameters)
```

像前面讨论的那样，其中m和n也是滤波区域的维数，'sliding' 表示处理过程是在输入图像 f 中逐个像素地滑动该$m \times n$区域，@fun引用一个函数，我们将该函数任意表示为fun，parameters

① A总有mn行，但列数可根据输入的尺寸变化，尺寸的选择由 colfilt 自动完成。

表示函数 fun 可能需要的参数（由逗号分隔开）。符号 @ 称为函数句柄，它是一种 MATLAB 数据类型，它包含有引用函数用到的一些信息。很快我们就会看到，这是一个非常有用的概念。

基于矩阵 A 的组织形式，函数 fun 必须分别对矩阵的每一列操作，并返回一个包含所有列的结果的行向量 v。v 的第 k 个元素表示的是对 A 中的第 k 列进行 fun 操作后的结果。因而，A 中可以有 MN 列，v 的最大维数为 $1 \times MN$。

上一节讨论的线性滤波方法已使用填充方法解决了空间滤波技术中固有的边界问题。但在使用 colfilt 时，在进行滤波之前，输入图像必须经过了填充。为此，我们可使用函数 padarray，对于二维函数，它的语法为

```
fp = padarray(f, [r c], method, direction)
```

其中，f 为输入图像，fp 为填充后的图像，[r c] 用于给出填充 f 的行数和列数，method 和 direction 的意义见表 3.3。例如，若 f = [1 2; 3 4]，则命令

```
>> fp = padarray(f, [3 2], 'replicate', 'post')
```

将产生结果

```
fp =
     1    2    2    2
     3    4    4    4
     3    4    4    4
     3    4    4    4
     3    4    4    4
```

若参量中不包括 direction，则默认值为 'both'。若参量中不包含 method，则默认用零来填充。若参量中不包括任何参数，则默认填充为零且默认方向为 'both'。在计算结束时，图像会被修剪为原始大小。

表 3.3　函数 padarray 的选项

选项	描述
方法	
'symmetric'	图像大小通过围绕边界进行镜像反射来扩展
'replicate'	图像大小通过复制外边界中的值来扩展
'circular'	图像大小通过将图像看成是一个二维周期函数的一个周期来进行扩展
方向	
'pre'	在每一维的第一个元素前填充
'post'	在每一维的最后一个元素后填充
'both'	在每一维的第一个元素前和最后一个元素后填充。此选项为默认值

例 3.8　使用函数 colfilt 实现非线性空间滤波

如函数 colfilt 示例的那样，现在我们来执行一个非线性滤波，该非线性滤波在任何点处的响应都是中心在该点的邻域内的像素亮度值的几何平均。大小为 $m \times n$ 的邻域中的几何平均是邻域内亮度值的乘积的 $1/mn$ 次幂。我们首先执行非线性滤波函数，调用 gmean:

```
function v = gmean(A)
mn = size(A, 1); % The length of the columns of A is always mn.
v = prod(A, 1).^(1/mn);
```

为削减边界效应，我们在函数 padarray 中使用 'replicate' 选项来填充输入图像：

```
>> f = padarray(f, [m n], 'replicate');
```

最后，我们调用函数 colfilt：

```
>> g = colfilt(f, [m n], 'sliding', @gmean);
```

此处有几个要点。首先，应当注意，尽管矩阵A是函数gmean中的一个参量，但它未包括在函数colfilt的参数中。这个矩阵可通过在函数colfilt中使用函数句柄自动传递给gmean。同样，由于矩阵A是由colfilt自动管理的，所以A中的列数是可变的（但像前面注释的那样，行数即列长总会是mn）。因此，函数colfilt每调用该参量一次，就要计算一次A的大小。在这种情况下，滤波的过程就是计算邻域内所有像素的乘积的 $1/mn$ 次幂。对于 (x, y) 的任意值，该点的滤波结果包含在v的适当列中。通过句柄@来识别的函数可是任何能从创建该函数句柄的位置调用的函数。关键要求是函数在A的列上进行操作，并返回一个行向量，该行向量包含了所有单独列的结果。然后，函数colfilt获得这些结果，对其重新排列，以便产生输出图像g。

一些常用的非线性滤波器可以通过其他MATLAB和IPT函数实现，如imfilter和ordfilt2（见3.5.2节）。例如，5.3节中的函数spfilt可以通过imfilter和MATLAB中的log和exp函数来实现几何平均滤波。当这些都能实现时，性能通常会更快，且使用的内存也只是函数colfilt所需内存的一小部分。然而，函数colfilt仍然是进行非线性滤波操作的最好选择，还没有可以取而代之的函数。

3.5 图像处理工具箱的标准空间滤波器

在这一节中，我们将讨论由IPT支持的线性和非线性滤波技术。其他有关非线性滤波实现的内容将在5.3节中介绍。

3.5.1 线性空间滤波器

工具箱支持一些预定义的二维线性空间滤波器，这些空间滤波器可通过使用函数fspecial来实现。用来生成滤波掩模w的函数fspecial的语法为

```
w = fspecial('type', parameters)
```

其中，'type'表示滤波器类型，'parameters'进一步定义了指定的滤波器。由函数fspecial支持的空间滤波器汇总于表3.4中，表中还提供了每一种滤波器的适用参数。

表3.4 函数 fspecial 支持的空间滤波器

类型	函数和参数
'average'	fspecial('average',[r c])。大小为r × c的一个矩形平均滤波器。默认值为3 × 3。若由一个数来代替[r c]，则表示方形滤波器
'disk'	fspecial('disk',r)。一个圆形平均滤波器（包含在2r + 1大小的正方形内），半径为r。默认半径为5
'gaussian'	fspecial('gaussian',[r c],sig)。一个大小为r × c的高斯低通滤波器，标准偏差为sig（正）。默认值为3 × 3和0.5。若由一个数来代替[r c]，则表示方形滤波器
'laplacian'	fspecial('laplacian',alpha)。一个大小为3 × 3的拉普拉斯滤波器，其形状由alpha指定，alpha是范围[0, 1]内的一个数。alpha的默认值为0.5
'log'	fspecial('log',[r c],sig)。一个大小为r × c的高斯-拉普拉斯（LoG）滤波器，标准偏差为sig（正）。默认值为5 × 5和0.5。若用一个数代替[r c]，则表示一个方形滤波器
'motion'	fspecial('motion',len,theta)。围绕一幅有着len个像素的图像线性运动时（就像照相机与景物的关系），输出一个滤波器。运动的方向为theta，其单位为度，即从水平方向逆时针转动的角度。默认值为9和0，表示沿水平方向9个像素的运动

该掩模可对增强结果进行精细的调整。但拉普拉斯算子的主要用途则基于上面刚刚讨论的两种掩模。现在，我们应用拉普拉斯算子来增强图3.16(a)所示的图像。这是一幅较为模糊的月球北极图像。在这种情况下，对图像的增强包括图像锐化，同时尽可能地保留其灰度色调。首先，我们生成并显示一个拉普拉斯滤波器：

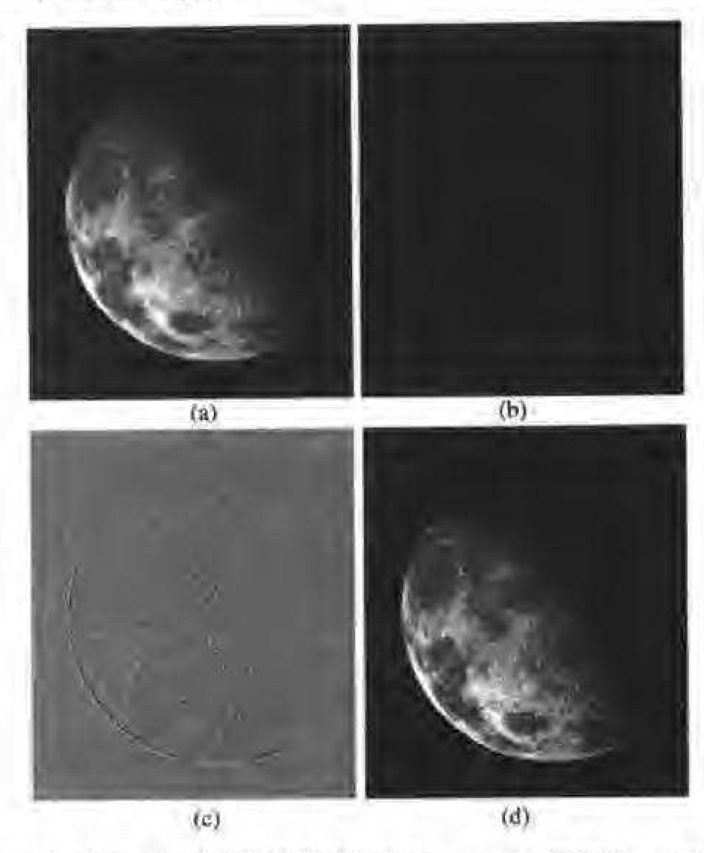


图3.16 (a)月球北极的图像；(b)经拉普拉斯滤波后的 uint8 类图像；(c)经拉普拉斯滤波后的 double类图像；(d)增强后的结果，即从(a)中减去(c)所得到的结果（原图像由NASA提供）

```
>> w = fspecial('laplacian', 0)
w =
    0.0000    1.0000    0.0000
    1.0000   -4.0000    1.0000
    0.0000    1.0000    0.0000
```

注意，该滤波器是 double 类滤波器，其 alpha 值为 0 的形状就是前面讨论过的拉普拉斯滤波器。我们可手工地将其形状指定为

```
>> w = [0 1 0; 1 -4, 1; 0 1 0];
```

下面我们将 w 应用到输入图像 f，该图像是 uint8 类图像：

```
>> g1 = imfilter(f, w, 'replicate');
>> imshow(g1, [ ])
```

图3.16(b)显示了结果图像。该结果看起来很合理，但存在一个问题：即所有像素都是正的。由于滤波器的中心系数为负，所以通常我们希望得到一个带负值的拉普拉斯图像。然而，在这种情况下f是 uint8 类图像，正如前一节讨论的那样，使用 imfilter 滤波后的输出图像与输入图像是同类图像，所以负值将被截掉。我们可以通过在滤波前将 f 转换为 double 类图像来解决这一问题：

```
>> f2 = im2double(f);
>> g2 = imfilter(f2, w, 'replicate');
>> imshow(g2, [ ])
```

示于图3.15(c)中的结果看起来要比使用拉普拉斯算子处理过的结果好一些。

最后，我们从原图像中减去用拉普拉斯算子处理过的结果，以还原失去的灰度色调(因为中心系数为负值)：

```
>> g = f2 - g2;
>> imshow(g)
```

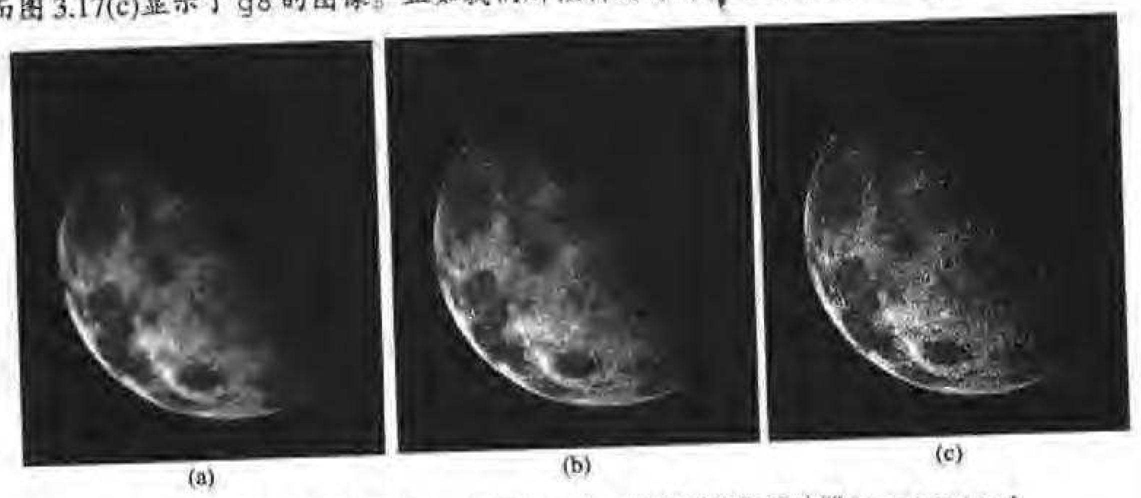
示于图3.16(d)中的结果要比原图像更清晰。

例3.10　手工指定滤波器和增强技术的比较

增强问题常常需要工具箱外的滤波器。拉普拉斯算子就是一个很好的例子。工具箱提供了一个大小为3×3的拉普拉斯滤波器，其中心为-4。通常，若希望得到更清晰的图像，则需要使用早些时候讨论过的中心为-8、其他值均为1的3×3拉普拉斯滤波器。本例的目的是用手工方法实现这个滤波器，并比较使用两种拉普拉斯方式得到的结果。命令序列如下所示：

```
>>  f = imread('moon.tif');
>>  w4 = fspecial('laplacian', 0);  % Same as w in Example 3.9.
>>  w8 = [1 1 1; 1 -8 1; 1 1 1];
>>  f = im2double(f);
>>  g4 = f - imfilter(f, w4, 'replicate');
>>  g8 = f - imfilter(f, w8, 'replicate');
>>  imshow(f)
>>  figure, imshow(g4)
>>  figure, imshow(g8)
```

为了便于比较，图3.17(a)再次显示了月球的原图像。3.17(b)是g4的图像，它与图3.16(d)相同，而图3.17(c)显示了g8的图像。正如我们所预料的那样，它要比图3.17(b)清晰得多。



(a)　(b)　(c)

图3.17　(a)月球北极的图像；(b)使用中心为-4的拉普拉斯滤波器'laplacian'增强后的图像；(c)使用中心为-8的拉普拉斯滤波器增强后的图像

3.5.2　非线性空间滤波器

IPT中常用于生成非线性空间滤波的一个工具是函数ordfilt2，它可以生成统计排序(order-statistic)滤波器(也称为排序滤波器，rank filter)。它们都是非线性空间滤波器，其响应基于对图

像邻域中所包含的像素进行排序，然后使用排序结果确定的值来替代邻域中的中心像素的值。本节的重点在于由函数 ordfilt2 生成的非线性滤波器。其他非线性滤波器将在 5.3 节中讨论。

函数 ordfilt2 的语法为

```
g = ordfilt2(f, order, domain)
```

该函数生成输出图像 g 的方式如下：使用邻域的一组排序元素中的第 order 个元素来替代 f 中的每个元素，而该邻域则由 domain 中的非零元素指定。这里，domain 是一个由 0 和 1 组成的大小为 $m \times n$ 的矩阵，该矩阵指定了将在计算中使用的邻域中的像素位置。在这种情况下，domain 的作用类似于掩模。计算中不使用对应于矩阵 domain 中的 0 的邻域中的像素。例如，要实现大小为 $m \times n$ 的最小滤波器，可使用语法

```
g = ordfilt2(f, 1, ones(m, n))
```

在该语句中，1 表示 mn 个样本中的第一个样本，ones(m,n) 创建了一个元素值为 1、大小为 $m \times n$ 的矩阵，表明邻域内的所有样本都将用于计算。

在统计学术语中，最小滤波器（一组排序元素中的第一个样本值）称为第 0 个百分位。同样，第 100 个百分位指的就是一组排序元素中的最后一个样本值，即第 mn 个样本。相应地，还有最大滤波器，它可使用语法

```
g = ordfilt2(f, m*n, ones(m, n))
```

来实现。

数字图像处理中最著名的统计排序滤波器是中值滤波器[①]，它对应的是第 50 个百分位。我们可以使用 MATLAB 函数 ordfilt2 来创建一个中值滤波器：

```
g = ordfilt2(f, median(1:m*n), ones(m, n))
```

其中 median(1:m*n) 简单地计算排序序列 1, 2, …, mn 的中值。函数 median 的通用语法为

```
v = median(A, dim)
```

其中，v 是向量，它的元素是 A 沿着维数 dim 的中值。例如，若 dim = 1，则 v 的每个元素就都是矩阵 A 中沿相应列的元素的中值。

基于实际应用中的重要性，工具箱提供了一个二维中值滤波函数：

```
g = medfilt2(f, [m n], padopt)
```

数组[m n] 定义了一个大小为 $m \times n$ 的邻域，中值就在该邻域上计算，而 padopt 指定了三个可能的边界填充选项之一：'zeros'（默认值）；'symmetric'，此地 f 按照镜像反射方式对称地沿其边界扩展；'indexed'，若 f 是 double 类图像，则以 1 来填充图像，否则以 0 来填充图像。该函数的默认形式为

```
g = medfilt2(f)
```

它使用一个大小为 3 × 3 的邻域来计算中值，并用 0 来填充输入图像的边界。

例 3.11 使用函数 medfilt2 进行中值滤波

中值滤波是降低图像椒盐噪声的有效工具。虽然我们将在第 5 章中详细介绍噪声的降低，但在此时简单介绍一下中值滤波的实现还是很有意义的。

① 回顾数值集合的中值 ξ，它是数值集中间位置的数，该数小于或等于 ξ，并且大于或等于 ξ。

图3.18(a)显示的是自动检测电路板时生成的X射线图像f，图3.18(b)显示的是带有椒盐噪声的同一幅图像，其中黑点和白点出现的概率均为0.2。这幅图像是使用函数imnoise生成的，5.2.1节将详细介绍该函数：

```
>> fn = imnoise(f, 'salt & pepper', 0.2);
```

图3.18(c)显示的是使用如下语句对有噪图像进行中值滤波处理后的图像：

```
>> gm = medfilt2(fn);
```

考虑图3.18(b)中的噪声程度，我们发现使用默认设置的中值滤波很好地降低了噪声。但在电路板的边界处出现了黑色的斑点，它们是由围绕图像的黑点导致的（回忆默认使用0来填充边界）。这种影响可通过使用'symmetric'选项来减弱：

```
>> gms = medfilt2(fn, 'symmetric');
```

图3.18(d)显示的结果近似于图3.18(c)中的结果，只是黑色边界效应不再明显。

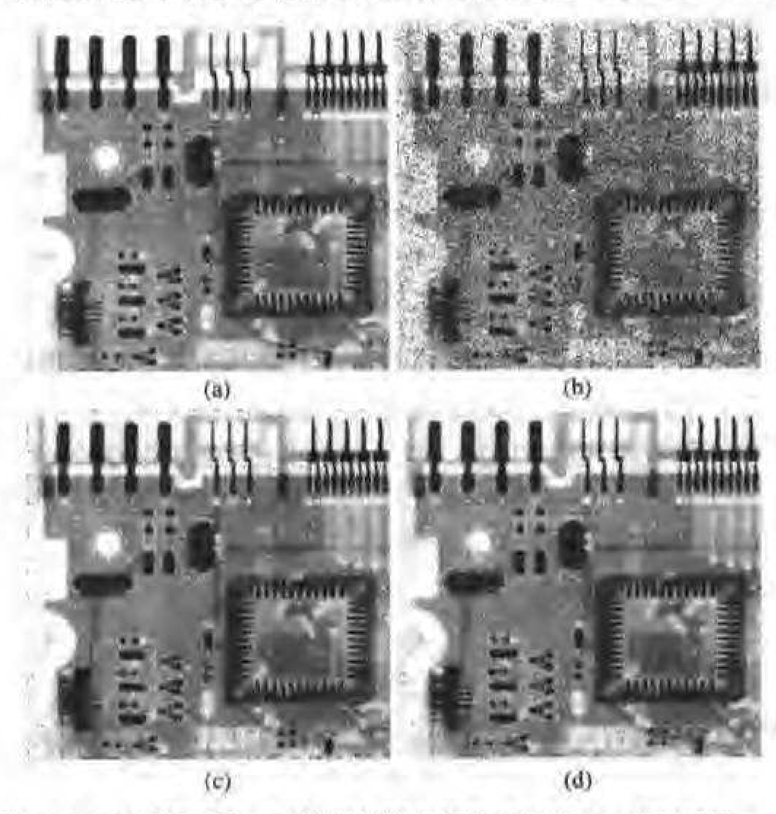


图3.18　中值滤波：(a)X射线图像；(b)被椒盐噪声污染的图像；(c)使用函数medfilt2的默认设置经中值滤波处理后的结果；(d)使用函数medfilt2的'symmetric'选项经中值滤波处理后的结果。注意(d)和(c)间的边界效应增强（原图像由Lixi公司提供）

小结

除图像增强处理外，本章中的内容是后续章节中许多主题的基础。例如，在第5章中我们还会遇到空间处理与图像还原，那时我们仍将仔细探讨MATLAB中的噪声消除和噪声生成函数。本章简要介绍过的一些空间掩模将会在第10章中广泛用于图像分割中的边缘检测。第4章将从频域的角度再次探讨卷积和相关的概念。从概念上讲，掩模处理和空间滤波的实现在本书中将多次提到。在后续章节中，我们将扩展本章中开始的讨论，并介绍在MATLAB中如何有效实现空间滤波器的其他内容。

第4章 频域处理

前言

本章的大部分内容与第3章中探讨的主题是并行的，只不过所有的滤波都是通过傅里叶变换在频域中实现的。除了是线性滤波的基础之外，傅里叶变换在图像增强、图像复原、图像数据压缩以及其他主要实际应用的设计和实现过程中都起着很重要的作用。在这一章中，我们重点关注的是如何在MATLAB中实现频域滤波。就像在第3章中一样，我们将举例说明在图像增强中的频域滤波，包括低通滤波、基本的高通滤波和高频强调滤波。我们还将简要说明结合空间域和频域进行图像处理的优点。本章涉及的概念和技术均很常见，这些内容的应用将在第5章、第8章和第11章中详细说明。

4.1 二维离散傅里叶变换

令$f(x, y)$表示一幅大小为$M \times N$的图像，其中$x = 0, 1, 2, \cdots, M-1$和$y = 0, 1, 2, \cdots, N-1$。f的二维离散傅里叶变换可表示为$F(u, v)$，如下式所示：

$$F(u, v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x, y)\mathrm{e}^{-j2\pi(ux/M+vy/N)}$$

其中$u = 0, 1, 2, \cdots, M-1$和$v = 0, 1, 2, \cdots, N-1$。我们可以将指数项扩展为正弦项和余弦项的形式，其中变量u和v用于确定它们的频率。频域系统是由$F(u, v)$所张成的坐标系，其中u和v用做（频率）变量。这类似于前一章所研究的空间域，空间域是由$f(x, y)$所张成的坐标系，其中x和y用做（空间）变量。由$u = 0, 1, 2, \cdots, M-1$和$v = 0, 1, 2, \cdots, N-1$定义的$M \times N$矩形区域常称为频率矩形。显然，频率矩形的大小与输入图像的大小相同。

离散傅里叶逆变换由下式给出：

$$f(x, y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u, v)\mathrm{e}^{j2\pi(ux/M+vy/N)}$$

其中$x = 0, 1, 2, \cdots, M-1$和$y = 0, 1, 2, \cdots, N-1$。因此，给定$F(u, v)$，我们就可借助于逆DFT得到$f(x, y)$。在这个等式中，$F(u, v)$的值有时称为傅里叶系数。

在DFT的一些公式中，$1/MN$项放置在正变换的前面，而在有些公式中则放在逆变换的前面。为了与MATLAB的实现形式相一致，本书中我们将该项放置于逆变换公式的前而，就像前而的公式中显示的那样。由于MATLAB中的数组索引是以1面不是以开头的，所以MATLAB中的`F(1, 1)`和`f(1,1)`相应于正变换和逆变换中的数学量$F(0, 0)$和$f(0, 0)$。

在频域原点处变换的值［如$F(0, 0)$］称为傅里叶变换的直流（dc）分量。该术语源于电气工程学，意指直流电（频率为零的电流）。不难看出，$F(0, 0)$等于$f(x, y)$的平均值的MN倍。

即使$f(x, y)$是实数，其变换通常也是复数。直观地分析一个变换的主要方法是计算它的频谱［即$F(u, v)$的幅度］，并将其显示为一幅图像。令$R(u, v)$和$I(u, v)$分别表示$F(u, v)$的实部和虚部，则傅里叶频谱定义为

$$|F(u, v)| = [R^2(u, v) + I^2(u, v)]^{1/2}$$

变换的相位角定义为

$$\phi(u,v) = \arctan\left[\frac{I(u,v)}{R(u,v)}\right]$$

使用我们熟悉的复数量的极坐标表示法，前两个函数可用于表示 $F(u, v)$：

$$F(u,v) = |F(u,v)|e^{-j\phi(u,v)}$$

功率谱定义为幅度的平方：

$$\begin{aligned} P(u,v) &= |F(u,v)|^2 \\ &= R^2(u,v) + I^2(u,v) \end{aligned}$$

为直观起见，我们是查看$|F(u,v)|$还是查看$P(u,v)$，这一点并不重要。

若$f(x, y)$是实数，则其傅里叶变换关于原点共轭对称，即

$$F(u,v) = F^*(-u,-v)$$

同样，傅里叶频谱也关于原点对称：

$$|F(u,v)| = |F(-u,-v)|$$

直接将它代入到$F(u, v)$的等式中可得

$$F(u,v) = F(u+M,v) = F(u,v+N) = F(u+M,v+N)$$

换言之，DFT在u和v方向上都是周期无穷的，周期由M和N决定。周期性也是DFT逆变换的一个重要属性：

$$f(x,y) = f(x+M,y) = f(x,y+N) = f(x+M,y+N)$$

换言之，傅里叶逆变换得到的图像也是周期无穷的。这是一个经常容易混淆的地方，因为一般情况下我们并不觉得由傅里叶逆变换得到的图像还需要转换为周期性的。我们可以简单地认为这是DFT及其逆变换的一个数学特性。还应牢记的是，DFT实现仅计算一个周期，所以我们可处理大小为$M \times N$的数组。

当我们考虑DFT数据与变换的周期的关系时，周期问题就会变得很重要。例如，图4.1(a)显示了一个一维变换$F(u)$的频谱。在这种情况下，周期性表示为$F(u) = F(u+M)$，同时它也满足$|F(u)| = |F(u+M)|$；另外，由对称性我们有$|F(u)| = |F(-u)|$。周期性表明$F(u)$的周期为M，而对称性表明变换的幅度集中在原点，如图4.1(a)所示。该图形和前面的描述表明，在原点以左的半个周期中，变换的幅度值从$M/2$到$M-1$重复。因为一维DFT仅在M个点上实现（如u的值在区间$[0, M-1]$内），所以在该区间计算机一维变换将产生两个紧邻的半周期。我们希望在区间$[0, M-1]$内得到一个完整且正确排序的周期。不难看出（见Gonzalez and Woods[2002]），期望的周期可以通过在计算变换前让$f(x)$乘以$(-1)^x$来得到。这样做可将变换的原点移动到点$u = M/2$，如图4.1(b)所示。现在，图4.1(b)中$u = 0$处的频谱值对应于图4.1(a)中的$|F(-M/2)|$。同样，图4.1(b)中$|F(M/2)|$和$|F(M-1)|$处的值对应于图4.1(a)中的$|F(0)|$和$|F(M/2-1)|$。

二维函数中同样也存在这个问题。现在计算该二维DFT将产生图4.2(a)所示矩形区间中的变换点，其中阴影区域表示使用本节开始定义的二维傅里叶变换公式计算所得到的$F(u, v)$的值。虚线矩形是周期性重复的，如图4.1(a)所示。阴影区域表示$F(u, v)$的值现在包含四个紧邻的四分之一周期，这四个四分之一周期交汇于图4.2(a)中显示的一个点。通过将原点的变换值移动到频率矩形的中心位置，可简化频谱的视觉分析。这可以通过在计算二维傅里叶变换之前将$f(x, y)$乘以$(-1)^{x+y}$来完成。周期然后会像图4.2(b)所示的那样排列。正如前面对一维变换的讨论一样，图4.2(b)中点$(M/2, N/2)$

处的频谱值与图4.2(a)中点(0, 0)处的频谱值相等，而图4.2(b)中点(0, 0)处的频谱值与图4.2(a)中点$(-M/2, -N/2)$处的频谱值相等。同样，图4.2(b)中点$(M-1, N-1)$处的频谱值等于图4.2(a)中点$(M/2-1, N/2-1)$处的频谱值。

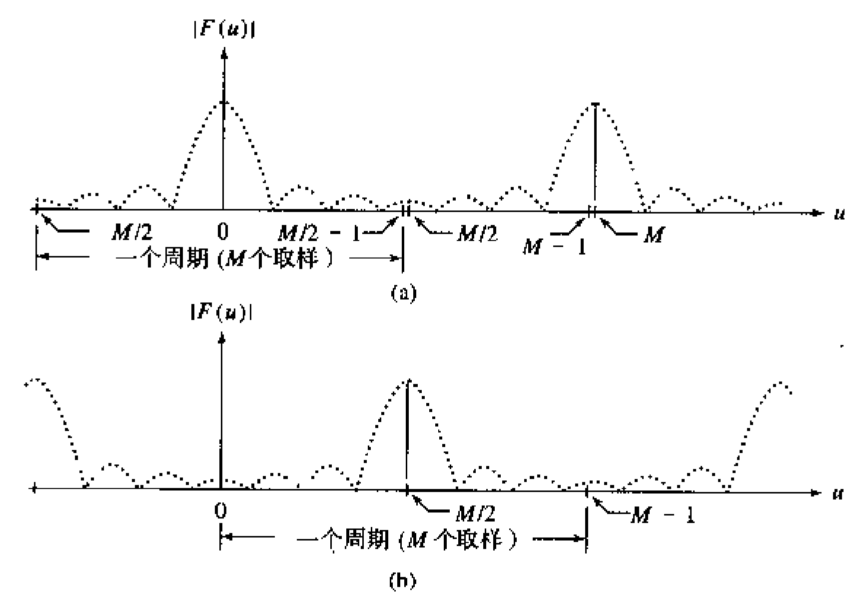


图4.1　(a)在区间[0, $M-1$]内显示紧邻的半个周期的傅里叶频谱；(b)在相同区间内，通过在计算傅里叶频谱之前用$f(x)$去乘以$(-1)^x$所得到的中心频谱

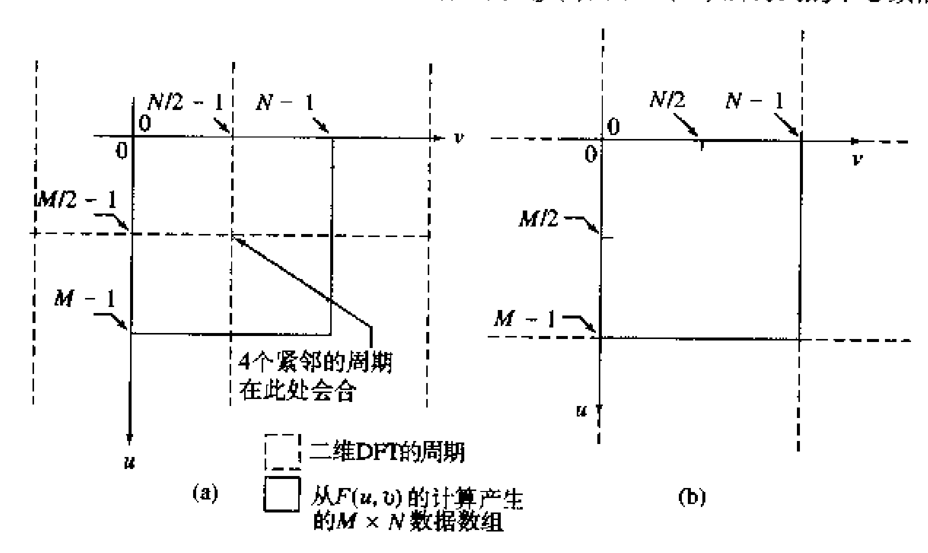


图4.2　(a)$M\times N$傅里叶频谱（阴影区域），显示了包含在频谱数据内的四个紧邻的四分之一周期；(b)在计算傅里叶变换之前，通过让$f(x, y)$乘以$(-1)^{x+y}$所得到的频谱。阴影区域仅显示了一个周期，因为这是执行$F(u, v)$的公式所得到的数据

前面关于将$f(x, y)$乘以$(-1)^{x+y}$来居中实现的讨论是一个非常重要的概念。在使用MATLAB进行处理时，计算变换不需要乘以$(-1)^{x+y}$，只需要使用函数`fftshift`重排数据即可。该函数及其用法将在下一节中讨论。

4.2　在MATLAB中计算并可视化二维DFT

在实际应用中，DFT及其逆变换可以通过使用快速傅里叶变换（FFT）算法来实现。一个大小为$M\times N$的图像数组`f`可以通过工具箱中的函数`fft2`得到，函数`fft2`的简单语法为

F = ff2(f)

该函数返回一个大小仍为$M \times N$的傅里叶变换，数据排列的形式如图4.2(a)所示；即数据的原点在左上角，而四个四分之一周期交汇于频率矩形的中心。

正如4.3.1节中解释的那样，使用傅里叶变换进行滤波时，需要对输入数据进行零填充。在这种情况下，语法变为

F = fft2(f, P, Q)

使用该语法，fft2将使用所要求的0的个数对输入图像进行填充，以便结果函数的大小为$P \times Q$。

傅里叶频谱可以使用函数abs来获得：

S = abs(F)

该函数计算数组的每一个元素的幅度（实部和虚部平方和的平方根）。

通过显示频谱的图像来进行可视化分析是频域处理的一个重要方面。例如，考虑一幅如图4.3(a)所示的简单图像f。我们计算它的傅里叶变换并使用如下步骤来显示其频谱：

```
>> F = fft2(f);
>> S = abs(F);
>> imshow(S, [ ])
```

图4.3(b)显示了结果。图像四个角上的亮点是前一节中提及的周期性导致的结果。

我们可以使用IPT函数fftshift将变换的原点移动到频率矩形的中心。该函数的语法为

Fc = fftshift(F)

其中，F是用fft2计算得到的变换，Fc是已居中的变换。函数fftshift通过交换F的象限来操作。例如，若a=[1 2;3 4]，则fftshift(a)=[4 3;2 1]。在变换计算后使用fftshift的结果，与在计算变换前将输入图像乘以$(-1)^{x+y}$所得到的结果相同。但要注意的是，这两种处理过程不可互换。换言之，若使用$\Im[\cdot]$表示一个参量的傅里叶变换，则$\Im[(-1)^{x+y}f(x,y)]$等于fftshift(fft2(f))，但不等于fft2(fftshift(f))。

在该例中键入

```
>> Fc = fftshift(F);
>> imshow(abs(Fc), [ ])
```

将产生如图4.3(c)所示的图像。居中后的结果在该图像中是很明显的。

虽然该移动像我们期望的那样完成了，但该频谱中值的动态范围（0到204 000）与8比特显示（此时中心处的明亮值占支配地位）相比要大得多。正如3.2.2节中讨论的那样，我们可以使用对数变换来处理该问题。因而，命令

```
>>S2 = log(1+abs(Fc));
>>imshow(S2, [ ])
```

产生了如图4.3(d)所示的结果。在这幅图中，可视细节的增加是很明显的。

函数ifftshift用于颠倒这种居中。该函数的语法为

F = ifftshift(Fc)

该函数也可用于将最初位于矩形中心的函数转换为中心位于矩形左上角的函数。我们将在4.4节中使用这一属性。

考虑居中问题时，要记住的是，若变量u和v的范围分别为0到$M-1$和0到$N-1$时，则频率矩形的中心位于$(M/2, N/2)$。例如，一个8×8频率矩形的中心点为(4, 4)，即每个坐标轴上从

4.3　频域滤波

从概念上讲，频域中的滤波是很简单的。在这一节中，我们将简单介绍频域滤波的概念以及如何使用MATLAB来实现频域滤波。

4.3.1　基本概念

空间域和频域线性滤波的基础都是卷积定理，该定理可以写为[1]

$$f(x, y) * h(h, y) \Leftrightarrow H(u, v)F(u, v)$$

和

$$f(x, y)h(h, y) \Leftrightarrow H(u, v) * G(u, v)$$

其中，符号"*"表示两个函数的卷积，双箭头两边的表达式组成了傅里叶变换对。例如，第一个表达式表明两个空间函数的卷积可以通过计算两个傅里叶变换函数的乘积的逆变换得到。相反地，两个空间函数的卷积的傅里叶变换恰好等于两个函数的傅里叶变换的乘积。同样的情况也出现在第二个表达式中。

在滤波问题上，我们更关注第一个表达式。空间域中的滤波由图像$f(x, y)$与滤波掩模$h(x, y)$组成。线性空间卷积曾在3.4.1节中做过介绍。根据卷积定理，我们可以在频域中通过让$F(u, v)$乘以$H(u, v)$来得到相同的结果，即空间滤波器的傅里叶变换。我们通常将$H(u, v)$称为滤波传递函数。

基本上，频域滤波的目的是选择一个滤波器传递函数，以便按照指定的方式修改$F(u, v)$。例如，图4.4(a)所示的滤波器有一个传递函数，在乘以一个居中处理后的函数$F(u, v)$后，该传递函数会衰减$F(u, v)$的高频分量，而保持低频分量相对不变。具有这种特性的滤波器称为低通滤波器。像在4.5.2节将会提到的那样，低通滤波器的结果会导致图像出现模糊现象（平滑）。图4.4(b)显示了经函数`fftshift`处理后的同一滤波器。在处理频率滤波时（此时输入图像的傅里叶变换未居中），这是本书中频繁使用的滤波器形式。

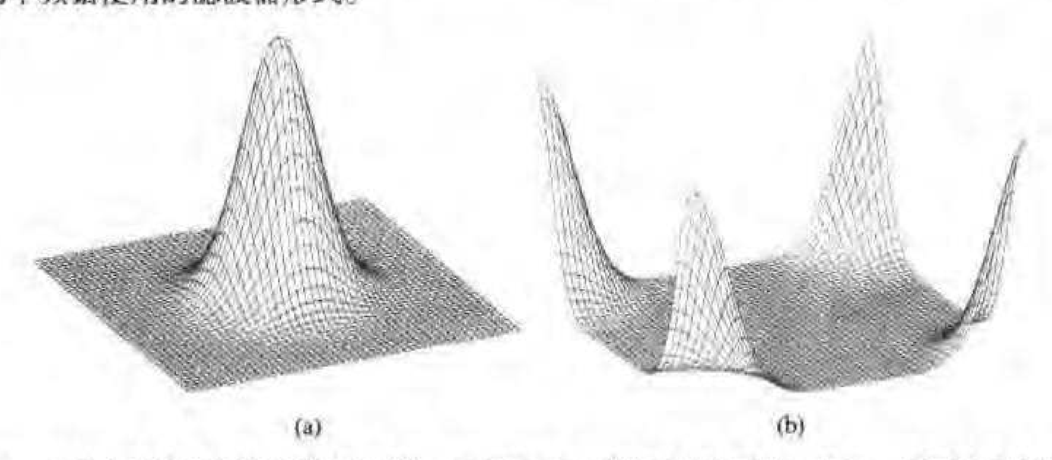


图4.4　(a)居中的低通滤波器的传递函数；(b)用于DFT滤波的传递函数。注意，它们是频域滤波器

基于卷积理论，我们知道为了在空间域中得到相应的滤波后的图像，仅需要计算积$H(u, v)F(u, v)$的傅里叶逆变换。应当记住，上述方法所得到的结果与我们在空间域中使用卷积所得到的结果是相同的，只要滤波掩模$h(x, y)$是$H(u, v)$的傅里叶逆变换。实际上，空间卷积常通过使用较小的掩模来简化，使用这种较小的掩模的目的是尽可能获得其频域对应内容的显著特性。

---

① 对于数据图像，仅当$f(x, y)$和$h(x, y)$正确地经过零填充后，表达式才严格有效，本节稍后将做介绍。

```
elseif nargin == 3
   m = max([AB CD]); % Maximum dimension.
   P = 2^nextpow2(2*m);
   PQ = [P, P];
else
   error('Wrong number of inputs.')
end
```

通过使用函数paddedsize计算的PQ，我们可使用函数fft2来计算经零填充后的FFT：

```
F = fft2(f, PQ(1), PQ(2))
```

该函数对f填充足够多的0，以便使结果图像的大小为PQ(1) × PQ(2)，然后像前面描述的那样计算FFT。注意，当在频域中使用填充时，滤波器函数的大小必须为PQ(1) × PQ(2)。

例4.1 使用填充和不使用填充的滤波效果

本例使用图4.5(a)所示的图像f来说明使用填充和不使用填充来进行滤波的区别。在下面的讨论中，我们使用函数lpfilter生成一个高斯低通滤波器［类似于图4.4(b)］，它带有指定值sigma(sig)。该函数将在4.5.2节中详细讨论，由于其语法很简单，所以我们在此处先使用它。下面的命令执行不使用填充的滤波：

```
>> [M, N] = size(f);
>> F = fft2(f);
>> sig = 10;
>> H = lpfilter('gaussian', M, N, sig);
>> G = H.*F;
>> g = real(ifft2(G));
>> imshow(g, [ ])
```

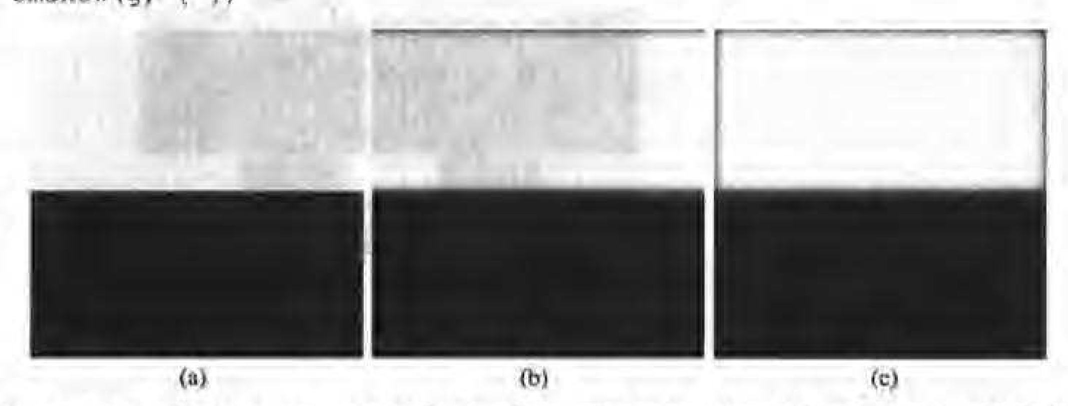


图4.5 (a)一幅大小为256 × 256的简单图像；(b)不使用填充的频域低通滤波处理后的图像；(c)使用填充的频域低通滤波处理后的图像。比较(b)和(c)中垂直边缘的明亮部分

图4.5(b)显示了图像g。像我们预计的那样，图像有些模糊，但垂直边缘并不模糊。其原因可通过图4.6(a)来解释，图4.6(a)显示了DFT计算中暗含的周期性。图像间的细白线使得图像更便于我们的查看，但它们并不是数据的一部分。虚线用于（任意）指定经fft2处理后的$M \times N$图像。想像一下对这个无限周期序列和一个模糊滤波器执行卷积操作的情况。很明显，当滤波器经过虚线图像的顶部时，会包含自己的一部分及其右上方的周期分量的底部。因而，当一块亮区和一块暗区在滤波器下面时，结果将是模糊且变暗的图像，如图4.5(b)的顶部所示。另一方面，当滤波器位于虚线图像的亮部时，它会遇到一个与周期分量相同的区域。因为常数区域的平均值仍是常数，所以在结果的这部分不会出现模糊现象。图4.5(b)所示图像的其他部分也可以用相同的方式加以解释。

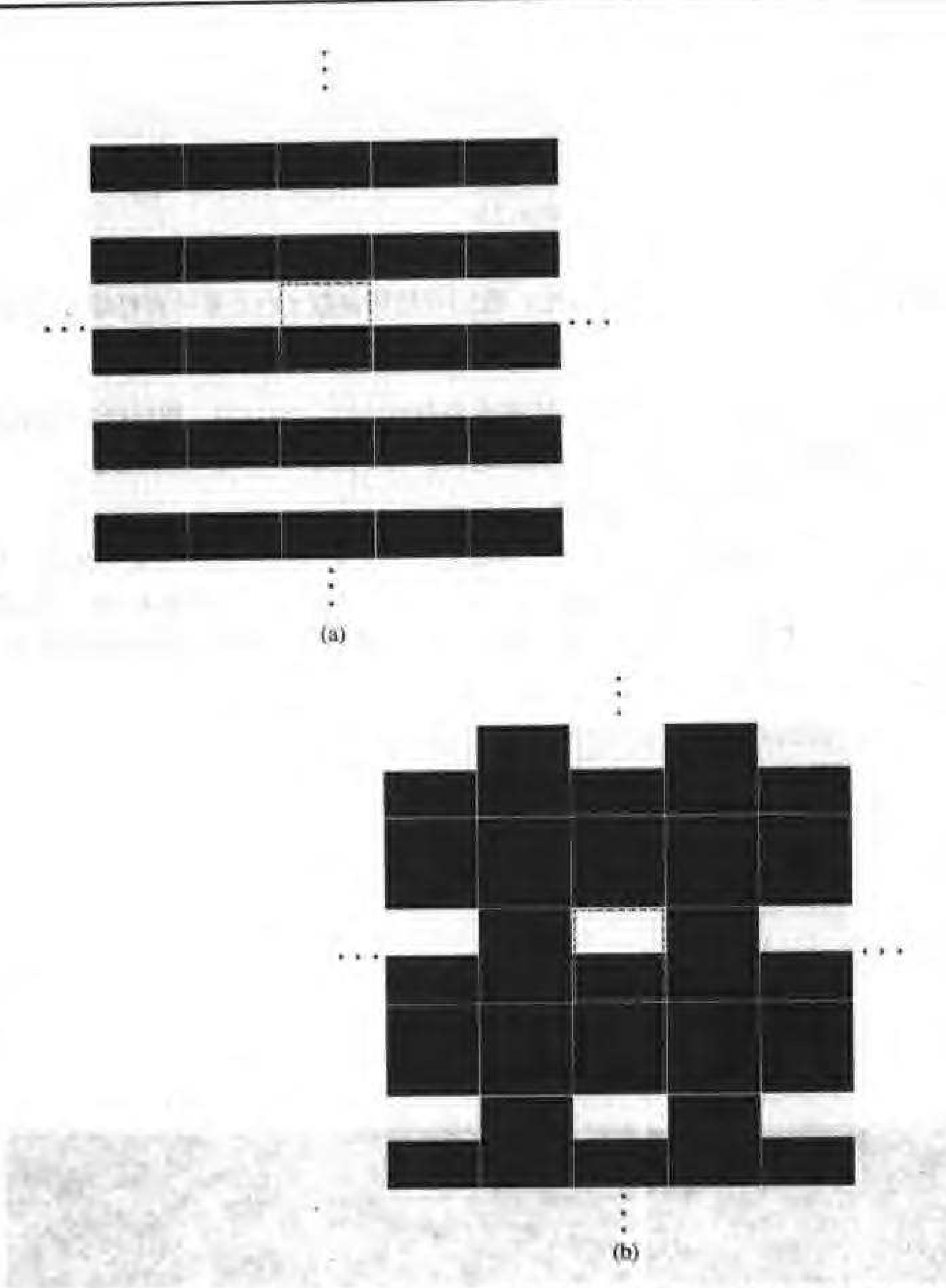


图 4.6 (a)图4.5(a)所示图像暗含的无限周期序列。虚线区域表示经fft2处理后的数据; (b)经零填充后的相同周期序列。两幅图像间的细白线只是为了方便我们的查看，它们并不是数据的一部分

下面考虑使用填充的滤波：

```
>> PQ = paddedsize(size(f));
>> Fp = fft2(f, PQ(1), PQ(2)); % Compute the FFT with padding.
>> Hp = lpfilter('gaussian', PQ(1), PQ(2), 2*sig);
>> Gp = Hp.*Fp;
>> gp = real(ifft2(Gp));
>> gpc = gp(1:size(f,1), 1:size(f,2));
>> imshow(gp, [ ])
```

这里我们使用了 2*sig，因为现在滤波器的大小是不使用填充时滤波器大小的两倍。

图 4.7 显示的是使用填充后的完整结果 gp。图 4.5(c)是通过将图 4.7 修剪为原图像大小而得到最终结果。该结果可借助于图 4.6(b)来解释，图 4.6(b)显示了经零填充后的虚线图像，零填充要在计算变换前于 fft2(f,PQ(1),PQ(2)) 中完成。暗含的周期性在前面已经介绍过。这幅图像现在具有均匀的黑色边界，因此，对这个无限序列和一个平滑滤波器执行卷积操作会在图像的亮区边缘产生较暗的模糊现象。执行如下的空间滤波可得到类似的结果：

```
>> h = fspecial('gaussian', 15, 7);
>> gs = imfilter(f, h);
```

回顾 3.4.1 节可知，调用函数 imfilter 时，默认情形下会使用 0 来填充图像的边界。



图 4.7　滤波后使用函数 ifft2 得到的经过填充后的图像。图像的大小为 512 × 512 像素

4.3.2　DFT 滤波的基本步骤

前几节的讨论可以概括为使用 MATLAB 函数的如下几个步骤，其中 f 是待滤波处理的图像，g 为处理结果，同时假设滤波函数 $H(u, v)$的大小与填充后的图像的大小相等。

1. 使用函数 paddedsize 获得填充参数：

```
PQ = paddedsize(size(f));
```

2. 得到使用填充的傅里叶变换：

```
F = fft2(f, PQ(1), PQ(2));
```

3. 使用本章讨论的任何方法，生成一个大小为 PQ(1) × PQ(2) 的滤波函数 H。该滤波函数的格式必须如图 4.4(b)所示。另外，若它如图 4.4(a)所示那样已居中，则在使用该滤波函数之前要令 H = fftshift(H)。

4. 将变换乘以滤波函数：

```
G = H.*F;
```

5. 获得 G 的傅里叶逆变换的实部：

```
g = real(ifft2(G));
```

6. 将左上部的矩形修剪为原始大小：

```
g = g(1:size(f, 1), 1:size(f, 2));
```

该滤波过程总结于图4.8中。预处理阶段包括确定图像大小、获得填充参数和生成一个滤波函数等步骤。后处理阶段包括计算结果的实部，修剪图像，以及将图像转换为uint8类图像或uint16类图像，以便于存储。

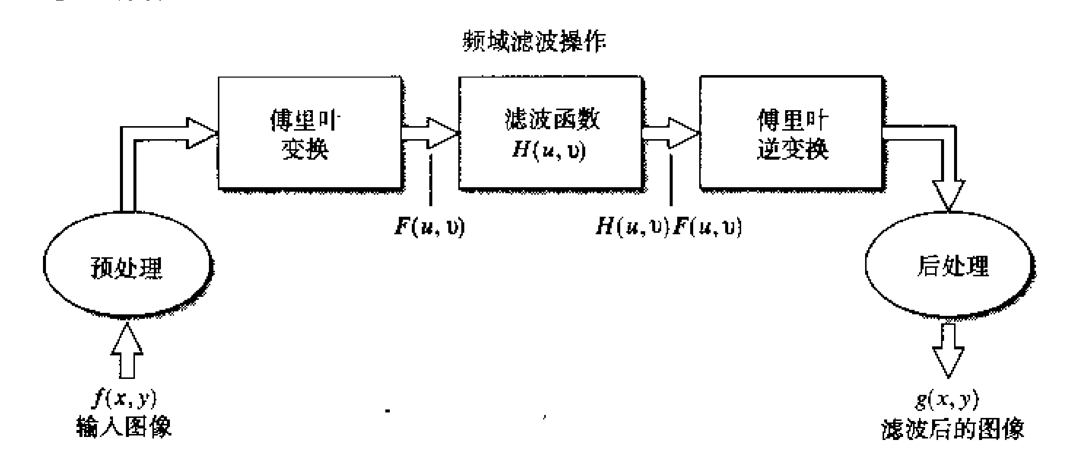


图4.8　频域滤波的基本步骤

图4.8中的滤波函数$H(u, v)$乘以$F(u, v)$的实部和虚部。若$H(u, v)$是实数，则结果的相位不变，这一点可以在相位公式中看到（见4.1节），但要注意的是，若实部和虚部的乘积相等，则它们都将被消去，相位角不变。以这种方式执行的滤波器被称为零相移滤波器，这种滤波器是本章中考虑的惟一线性滤波器。

由线性系统理论可知，在某种适度条件下，输入到线性系统的一个冲击完全可以表征系统。当我们按本书中的方法处理有限的离散数据时，线性系统的响应（包括对冲击的响应）也是有限的。若线性系统仅是一个空间滤波器，则通过简单地观察它对冲击的响应，我们就可完全确定该滤波器。通过这种方式确定的滤波器称为有限冲击响应（FIR）滤波器。本书中提到的所有线性空间滤波器都是FIR滤波器。

4.3.3　用于频域滤波的M函数

上一节中介绍的滤波步骤在本章及后续几章中都会用到，因此，我们希望得到这样一个M函数：该函数可接受输入图像和一个滤波函数，可处理所有的滤波细节并输出经滤波和剪切后的图像。下面的函数可实现这些工作。

```
function g = dftfilt(f, H)
%DFTFILT Performs frequency domain filtering.
%   G = DFTFILT(F, H) filters F in the frequency domain using the
%   filter transfer function H. The output, G, is the filtered
%   image, which has the same size as F. DFTFILT automatically pads
%   F to be the same size as H. Function PADDEDSIZE can be used
%   to determine an appropriate size for H.
%
%   DFTFILT assumes that F is real and that H is a real, uncentered,
%   circularly-symmetric filter function.

% Obtain the FFT of the padded input.
F = fft2(f, size(H, 1), size(H, 2));

% Perform filtering.
```

```
g = real(ifft2(H.*F));
% Crop to original size.
g = g(1:size(f, 1), 1:size(f, 2));
```

生成频域滤波器的技术将在下面三节中讨论。

4.4 从空间滤波器获得频域滤波器

通常情况下，当滤波器较小时，空间域滤波要比频域滤波更有效。"小"的定义较为复杂，取决于众多因素，如所使用的机器和算法、缓冲器的大小、所处理数据的复杂度等。Brigham[1988]使用一维函数进行的比较表明，使用FFT算法的滤波要快于空间滤波，此时函数有按顺序排列的32个点，但这个数字并不大。为得到两种方法间的有意义比较，知道如何将一个空间滤波器转换为同等的频域滤波器是很有用的。

相对于一个给定的空间滤波器h，生成一个频域滤波器H的明显方法是令H = fft2(h,PQ(1),PQ(2))，其中向量PQ的值取决于我们想要对其滤波的图像的大小，如上一节讨论的那样。但在本节中，我们感兴趣的是如下两个主题：(1)如何将空间滤波器转换为等价的频域滤波器；(2)如何比较使用函数imfilter的空间滤波与使用上一节所述方法的频域滤波。正如3.4.1节中分析的那样，函数imfilter使用了相关，且滤波函数的原点在中心处，为了使这两种方法等同，我们需要预处理大量数据。工具箱提供了函数freqz2，它可以完成这些工作并输出频域中的相应滤波器。

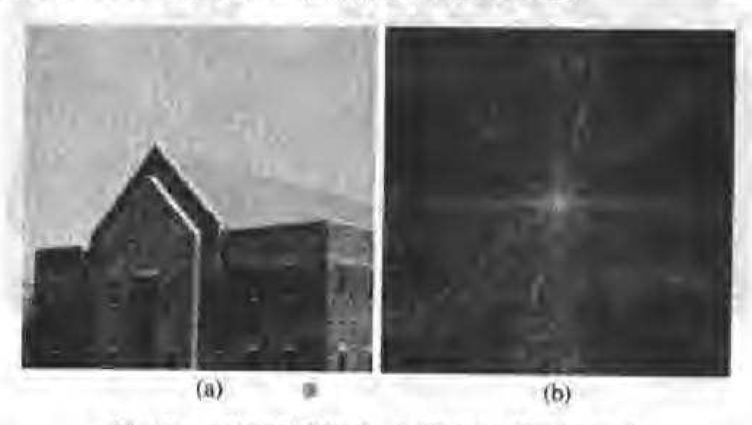
函数freqz2用于计算FIR滤波器的频率响应，如4.3.2节末尾提到的那样，这是本书中惟一考虑的线性滤波器。结果是我们所希望的频域滤波器。适用于当前讨论的语法为

```
H = freqz2(h, R, C)
```

其中，h是一个二维空间滤波器，H是相应的二维频域滤波器，R为H的行数，C为H的列数。通常，如4.3.1节中说明的那样，我们仅R = PQ(1)，C = PQ(2)。若freqz2无输出参量，则H的绝对值会在MATLAB桌面上显示为三维透视图。函数freqz2的使用方法可通过一个例子来解释。

例4.2 空间域滤波与频域滤波的比较

考虑图4.9(a)所示的图像f，其大小为600 × 600像素。接下来，我们生成频域滤波器H，它相关于增强垂直边缘的Sobel空间滤波器（见表3.4）。然后，我们比较在空间域中采用Sobel掩模（使用函数imfilter）对f滤波的结果与在频域中进行等价操作的结果。实际上，正如早些时候提到的那样，使用类似Sobel这样的小型滤波器所进行的滤波可直接在空间域中完成。之所以选择这个滤波器来表明我们的观点，是因为它的系数简单且滤波的结果很直观，从而便于我们的比较。较大的空间滤波器也可按相同的方式处理。



(a) (b)

图4.9 (a)灰度级图像；(b)图像的傅里叶频谱

图4.9(b)是图像f的傅里叶频谱，它可通过下列方法获得：

```
>> F = fft2(f);
>> S = fftshift(log(1 + abs(F)));
>> S = gscale(S);
>> imshow(S)
```

接着，我们使用函数 fspecial 来生成空间滤波器：

```
h = fspecial('sobel')'
h =
     1     0    -1
     2     0    -2
     1     0    -1
```

为了查看相应频域滤波器的图形，我们键入

```
>> freqz2(h)
```

图4.10(a)显示了无坐标轴的结果（获得透视图的方法将在4.5.3节中讨论）。滤波器本身可使用如下命令获得：

```
>> PQ = paddedsize(size(f));
>> H = freqz2(h, PQ(1), PQ(2));
>> H1 = ifftshift(H);
```

如前所述，ifftshift 需要重排数据序列，以便使得原点位于频率矩形的左上角。图4.10(b)显示了 abs(H1) 的图形。图4.10(c)和图4.10(d)以图像的形式显示了 H 和 H1 的绝对值，所用命令为

```
>> imshow(abs(H), [ ])
>> figure, imshow(abs(H1), [ ])
```

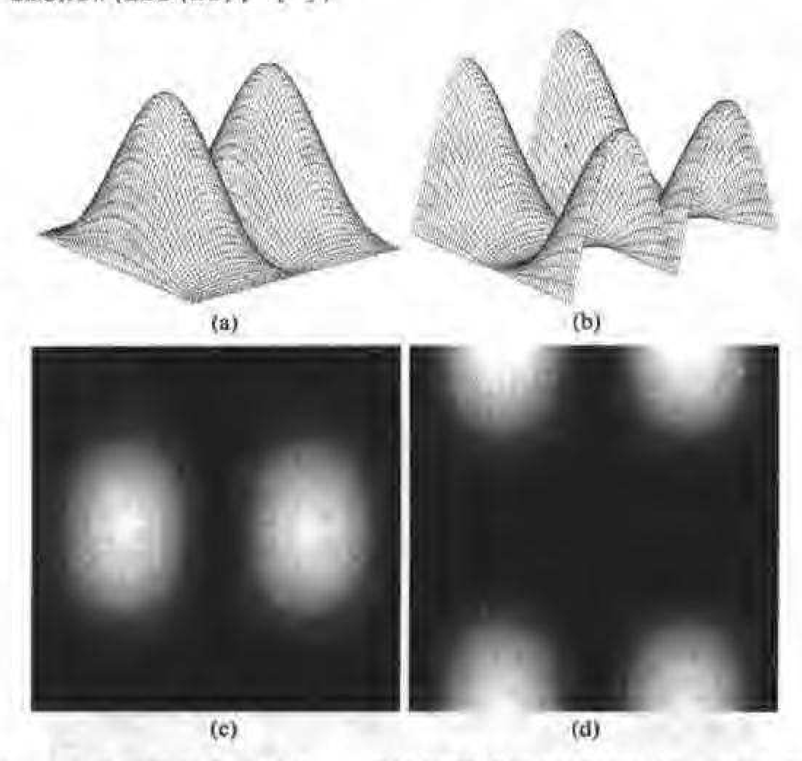


图4.10　(a)相应于垂直Soble掩模的频域滤波器的绝对值；(b)使用函数 fftshift处理后的同一滤波器。图(c)和(d)是以图像形式显示的滤波器

接着，我们生成滤波后的图像。在空间域中，我们使用

```
>> gs = imfilter(double(f), h);
```

该语句默认采用0进行边界填充。使用语句

```
gf = dftfilt(f, H1);
```

给出的频域处理，可得到滤波后的图像。

图4.11(a)和图4.11(b)显示了执行命令

```
>> imshow(gs, [ ])
>> figure, imshow(gf, [ ])
```

后的结果。

图像中的灰色调是由于gs和gf存在负值，这会使得图像的平均值在使用命令imshow后增大。如6.6.1节和10.1.3节中讨论的那样，前面生成的Sobel掩模h使用响应的绝对值来检测图像的垂直边缘。这样，显示刚计算过的图像的绝对值就更加方便了。图4.11(c)和图4.11(d)显示了使用命令

```
>> figure, imshow(abs(gs), [ ])
>> figure, imshow(abs(gf), [ ])
```

得到的图像。

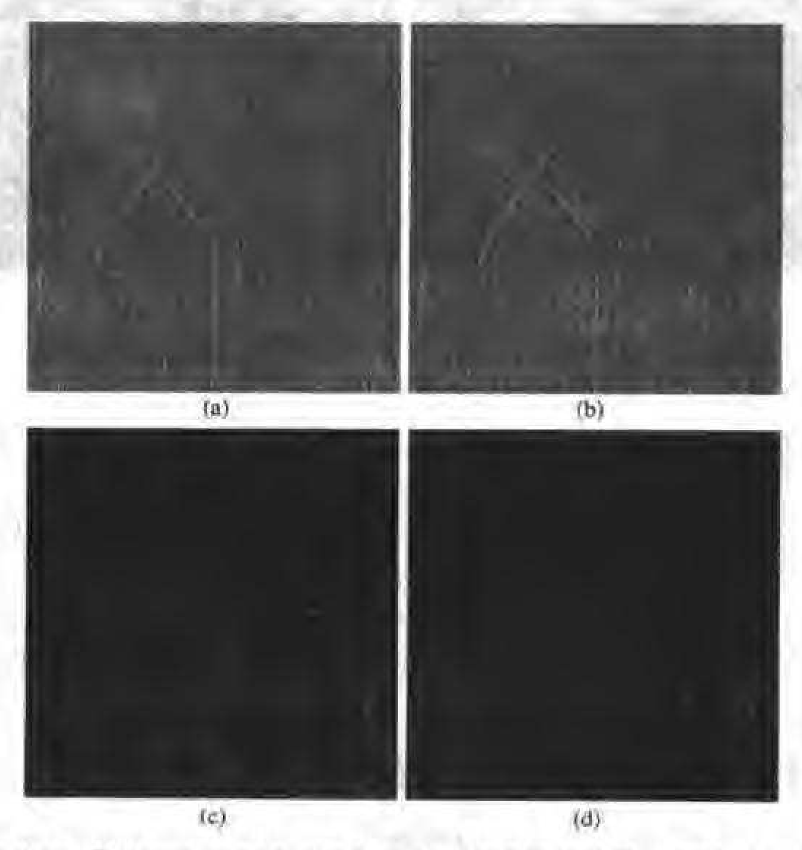


图4.11 (a)在空间域使用垂直Soble掩模对图4.9(a)滤波后的结果；(b)在频率使用图4.10(b)所示滤波器对同一图像滤波后的结果；图(c)和图(d)分别是图(a)和图(b)的绝对值

通过创建一幅阈值二值图像，我们可更清楚地看到边缘：

```
>> figure, imshow(abs(gs) > 0.2*abs(max(gs(:))))
>> figure, imshow(abs(gf) > 0.2*abs(max(gf(:))))
```

其中，选择乘数0.2的目的是仅显示强度比gs和gf的最大值的20%还要大的边缘。图4.12(a)和图4.12(b)显示了结果。

使用空间域滤波和频域滤波得到的图像对所有实用目的来说，都是相同的。下面我们通过计算它们的关来确证这一点：

```
>> d = abs(gs - gf);
```

最大差和最小差别分别为

```
>> max(d(:))
ans =
    5.4015e-012
>> min(d(:))
ans =
    0
```

刚刚解释的方法可在频域中实现3.4.1节和3.5.1节中介绍的空间域滤波方法，也可实现其他任意大小的FIR空间滤波器。

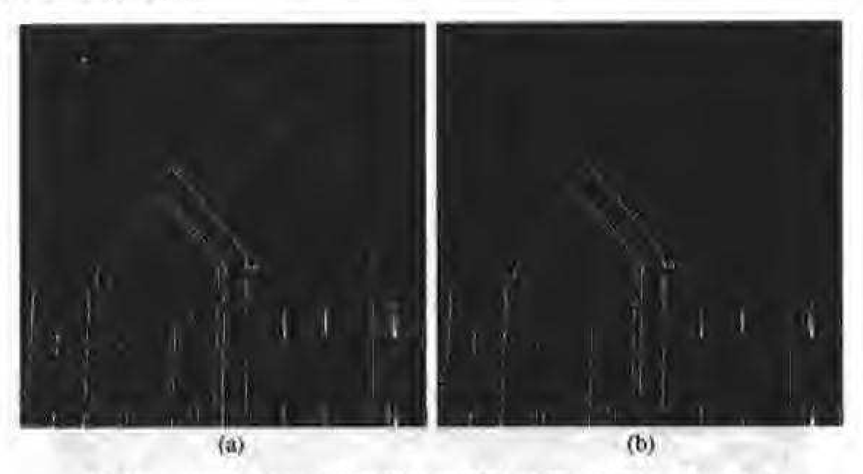


图4.12 (a)图4.11(c)经阈值处理后的结果；(b)图4.11(d)经阈值处理后的结果。两者用来更清楚地显示主要边缘

4.5 在频域中直接生成滤波器

在这一节中，我们将介绍如何在频域中直接生成滤波器。我们关注的是对称滤波器，即那些指定为至变换的原点的距离的各种函数的滤波器。用来实现这些滤波器的M函数仅是一些基本函数，它们可以很容易推广到具有相同结构的其他函数中。下面，我们将实现几个著名的平滑（低通）滤波器，然后介绍如何用MATLAB的线框图和表面图来可视化滤波器。最后，我们将简单地介绍锐化（高通）滤波器。

4.5.1 建立用于实现频域滤波器的网格数组

在下面的讨论中，M函数的核心是需要在频率矩形中计算任意点到指定点的距离函数。在MATLAB中，由于FFT的计算假设变换的原点在频率矩形的左上角，所以我们的距离计算也要用到那个点。数据可通过使用函数fftshift来重新排序，以实现可视化（因而原点的值被转换为频率矩形的中心）。

下面的M函数dftuv，提供了距离计算及其他类似应用所需要的网格数组（函数meshgrid用于后续代码中的解释，请参阅2.10.4节）。由函数dftuv生成的网格数组已满足使用fft2和ifft2的处理的需要，因而无须重排数据。

```
function [U, V] = dftuv(M, N)
%DFTUV Computes meshgrid frequency matrices.
%   [U, V] = DFTUV(M, N) computes meshgrid frequency matrices U and
%   V.  U and V are useful for computing frequency-domain filter
%   functions that can be used with DFTFILT.  U and V are both
%   M-by-N.

% Set up range of variables.
u = 0:(M - 1);
v = 0:(N - 1);

% Compute the indices for use in meshgrid.
idx = find(u > M/2);
u(idx) = u(idx) - M;
idy = find(v > N/2);
v(idy) = v(idy) - N;

% Compute the meshgrid arrays.
[V, U] = meshgrid(v, u);
```

例4.3 使用函数dftuv

下面的命令用于在大小为8 × 5的矩形中计算每一点到矩形原点的距离平方。

```
>> [U, V] = dftuv(8, 5);
>> D = U.^2 + V.^2
D =
     0     1     4     4     1
     1     2     5     5     2
     4     5     8     8     5
     9    10    13    13    10
    16    17    20    20    17
     9    10    13    13    10
     4     5     8     8     5
     1     2     5     5     2
```

注意，该距离在左上角处为0，最大距离的位置是频率矩形的中心，并遵循图4.2(a)中解释的基本格式。我们可以使用函数fftshift来获得相对于频率矩形中心的距离，

```
>> fftshift(D)
ans =
    20    17    16    17    20
    13    10     9    10    13
     8     5     4     5     8
     5     2     1     2     5
     4     1     0     1     4
     5     2     1     2     5
     8     5     4     5     8
    13    10     9    10    13
```

距离为0的点的坐标为(5, 3)，且数组关于该点对称。

4.5.2 低通频域滤波器

理想低通滤波器（ILPF）具有传递函数：

$$H(u,v)=\begin{cases}1 & 若\ D(u,v)\leqslant D_0\\ 0 & 若\ D(u,v)>D_0\end{cases}$$

其中，D_0为指定的非负数，$D(u, v)$为点(u, v)到滤波器中心的距离。$D(u, v) = D_0$的点的轨迹为一个圆。注意，若滤波器H乘以一幅图像的傅里叶变换，则我们会发现理想滤波器切断（乘以0）了圆外F的所有分量，而圆上和圆内的点不变（乘以1）。虽然这个滤波器不能使电子元件来模拟实现，但它可以在计算机中用前面介绍的传递函数来实现。理想滤波器通常用来解释如折叠误差等的问题。

n阶巴特沃兹低通滤波器（BLPF）（在距离原点D_0处出现截止频率）的传递函数为

$$H(u,v)=\frac{1}{1+[D(u,v)/D_0]^{2n}}$$

与理想低通滤波器不同的是，巴特沃兹低通滤波器的传递函数并不是在D_0处突然不连续。对于具有平滑传递函数的滤波器，我们通常要定义一个截止频率，在该点处$H(u, v)$会降低为其最大值的某个给定比例。在前面的等式中，当$D(u, v) = D_0$时，$H(u, v) = 0.5$（降为其最大值1的50%）。

高斯低通滤波器（GLPF）的传递函数为

$$H(u,v)=\mathrm{e}^{-D^2(u,v)/2\sigma^2}$$

其中，σ为标准偏差。通过令$\sigma = D_0$，我们可以根据截止参数D_0得到表达式

$$H(u,v)=\mathrm{e}^{-D^2(u,v)/2D_0^2}$$

当$D(u, v) = D_0$时，滤波器由其最大值1降为0.607。

例 4.4 低通滤波器

作为一个示例，我们对图4.13(a)所示的、大小为500 × 500像素的图像f应用一个高斯低通滤波器。我们使用的D_0值等于填充后的图像的宽度的5%。根据4.3.2节中介绍的滤波步骤，我们有

```
>> PQ = paddedsize(size(f));
>> [U, V] = dftuv(PQ(1), PQ(2));
>> D0 = 0.05*PQ(2);
>> F = fft2(f, PQ(1), PQ(2));
>> H = exp(-(U.^2 + V.^2)/(2*(D0^2)));
>> g = dftfilt(f, H);
```

键入

```
>> figure, imshow(fftshift(H), [ ])
```

我们可以图像的形式查看该滤波器［见图4.13(b)］。

键入

```
>> figure, imshow(log(1 + abs(fftshift(F))), [ ])
```

可将频谱显示为一幅图像［见图4.13(c)］。

最后，使用命令

```
>> figure, imshow(g, [ ])
```

可显示输出图像，如图4.13(d)所示。

正如我们所预料的那样，这幅图像与原图像相比，要模糊一些。

```
switch type
case 'ideal'
   H = double(D <= D0);
case 'btw'
   if nargin == 4
      n = 1;
   end
   H = 1./(1 + (D./D0).^(2*n));
case 'gaussian'
   H = exp(-(D.^2)./(2*(D0^2)));
otherwise
   error('Unknown filter type.')
end
```

4.6 节中还将使用函数 `lpfilter` 来和成高通滤波器。

4.5.3 线框图与表面图

3.3.1 节中已介绍了一个变量的函数的图形。在下面的讨论中，我们将介绍三维线框图及表面图，这些图形可用于可视化二维滤波器的传递函数。对于给定的二维函数 `H`，绘制线框图的最简方法是使用函数 `mesh`，该函数的语法为

```
mesh(H)
```

该函数将绘制一个 `x = 1:M` 和 `y = 1:N` 的线框图，其中 `[M,N] = size(H)`。若 `M` 和 `N` 很大，则线框图的密码会大到不可接受，在这种情况下，我们可以使用语法

```
mesh(H(1:k:end, 1:k:end))
```

来绘制第 `k` 个点。

经验表明，沿每个轴 40 到 60 个细分可在外观与分辨率之间提供了较好的平衡。

MATLAB 默认情况下会使用彩色来绘制网线。命令

```
colormap([0 0 0 ])
```

可将线框设置为黑色（我们将在第 6 章中讨论函数 `colormap`）。MATLAB 还可在网线图上添加网格和坐标轴。使用命令

```
gird off
axis off
```

可以关闭网格和坐标轴。在上面两条语句中通过将 `off` 更改为 `on`，可打开网格和坐标轴。最后，查看点（观测者的位置）由函数 `view` 控制，该函数的语法为

```
view(az, el)
```

如图 4.14 所示，`az` 和 `el` 分别代表方位角和仰角（度数）。箭头表示正方向。默认值是 `az = -37.5` 和 `el = 30`，在图 4.14 中，该值将观测者置于由 $-x$ 轴和 $-y$ 轴定义的象限内，以便观测由 x 轴和 y 轴定义的象限。

要确定当前视图的几何结构，可键入

```
>>[az, el] = view;
```

要将查看点设置为默认值，可键入

```
>>view(3)
```

单击图形窗口工具条上的 **Rotate 3D** 按钮，然后单击并拖动图形窗口，可交互式地修改查看点。

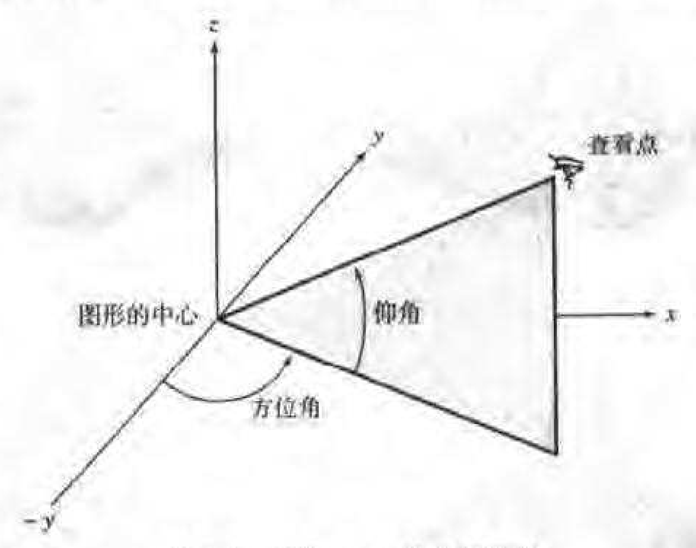


图 4.14 函数 view 的几何结构

如第6章中讨论的那样，我们可以在笛卡儿坐标(x, y, z)中指定观测者的位置，在处理 RGB 数据时，笛卡儿坐标是理想的坐标。然而，对于用于普通查看目的的图形，刚才讨论的方法仅包含两个参数，因而更加直观。

例 4.5 绘制线框图

考虑一个类似于例 4.4 中使用的高斯低通滤波器：

```
>> H = fftshift(lpfilter('gaussian', 500, 500, 50));
```

图 4.15(a)显示了使用如下命令产生的线框图：

```
>> mesh(H(1:10:500, 1:10:500))
>> axis([0 50 0 50 0 1])
```

命令 axis 已在 3.3.1 节中描述过，只是它现在还包含有 z 轴。

正如我们在本节前面提到的那样，线框图默认为彩色，它从底部的蓝色渐变到顶部的红色。键入命令

```
>> colormap([0 0 0])
>> axis off
>> grid off
```

我们可将图中的线条由彩色转换成黑色，并消除轴和网格。图 4.15(b)显示了其结果。图 4.15(c)显示了命令

```
>> view(-25, 30)
```

的结果，该命令使观测者稍微向右移一些，但保持仰角不变。最后，图 4.15(d)显示了保持方位角为 −25 并将仰角设置为 0 后所得的结果：

```
>> view(-25, 0)
```

这个例子展示了 mesh 这个简单函数的有效绘图能力。

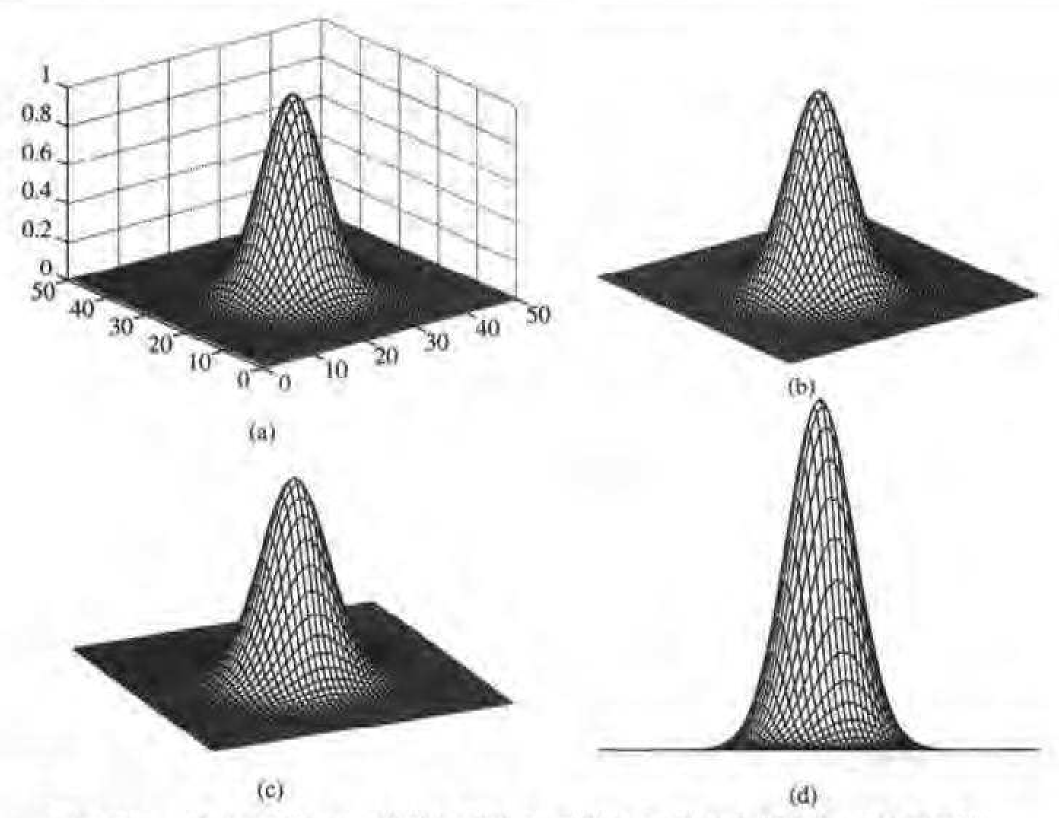


图 4.15 (a)使用函数mesh得到的图形；(b)删除轴和网格后的图形；(c)使用函数view得到的一幅不同的透视图；(d)使用同一函数得到的另一幅视图

有时，需要将一个函数绘制成表面图来代替线框图。这时可使用函数surf，其基本语法为

```
surf(H)
```

除了网格中的四边形使用彩色填充（称为小面描影）外，该函数产生的图形与函数mesh产生的图形相同。要将彩色转换成灰色，我们可以使用命令

```
colormap(gray)
```

函数axis、函数grid和函数view的工作方式类似于前面提到的函数mesh。例如，图4.16(a)是如下命令序列产生的结果：

```
>> H = fftshift(lpfilter('gaussian', 500, 500, 50));
>> surf(H(1:10:500, 1:10:500))
>> axis([0 50 0 50 0 1])
>> colormap(gray)
>> grid off; axis off
```

使用命令

```
shading interp
```

进行插值，可平滑小面描影并删除网格线。在提示符下键入该命令，可产生图4.16(b)所示的结果。

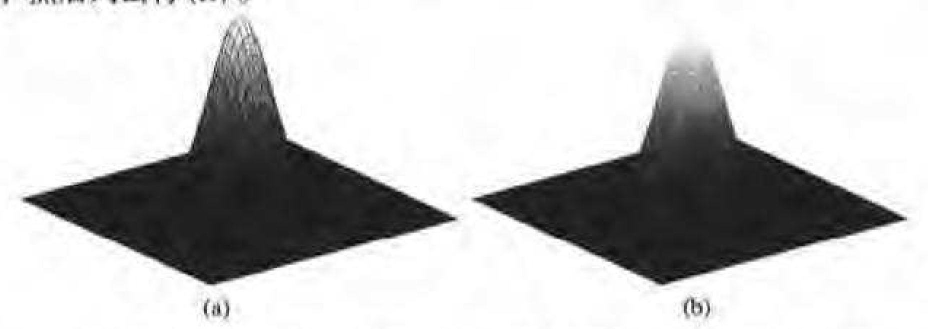
若目标是绘制两个变量的解析函数，则可使用函数meshgrid产生坐标值，并由这些坐标值产生在mesh或surf中使用的离散（取样）矩阵。例如，若要绘制函数

$$f(x, y) = xe^{(-x^2-y^2)}$$

其x轴和y轴都是从 -2 到 2，增量为0.1，则可以写出语句

```
>> [Y, X] = meshgrid(-2:0.1:2, -2:0.1:2);
>> Z = X.*exp(-X.^2 - Y.^2);
```

然后，像先前一样使用mesh(Z)或surf(Z)。回顾2.10.4节的讨论可知，函数meshgrid中将首先列出列(Y)，然后列出行(X)。



(a) (b)

图4.16 (a)使用函数surf得到的图形；(b)使用命令shading interp得到的结果

4.6 锐化频域滤波器

就像低通滤波会使图像变得模糊那样，高通滤波通过削弱傅里叶变换的低频而保持高频相对不变，会使得图像变得更加清晰（锐化）。在这一节中，我们将考虑几种高通滤波的方法。

4.6.1 基本的高通滤波器

给定一个低通滤波器的传递函数$H_{lp}(u,v)$，通过使用如下的简单关系，我们可以获得相应高通滤波器的传递函数：

$$H_{hp}(u,v) = 1 - H_{lp}(u,v)$$

因此，上一节中详述的函数lpfilter稍加改动，就可变成高通滤波器，如下所示：

```
function H = hpfilter(type, M, N, D0, n)
%HPFILTER Computes frequency domain highpass filters.
%   H = HPFILTER(TYPE, M, N, D0, n) creates the transfer function of
%   a highpass filter, H, of the specified TYPE and size (M-by-N).
%   Valid values for TYPE, D0, and n are:
%
%   'ideal'    Ideal highpass filter with cutoff frequency D0. n
%              need not be supplied. D0 must be positive.
%
%   'btw'      Butterworth highpass filter of order n, and cutoff
%              D0. The default value for n is 1.0. D0 must be
%              positive.
%
%   'gaussian' Gaussian highpass filter with cutoff (standard
%              deviation) D0. n need not be supplied. D0 must be
%              positive.
% The transfer function Hhp of a highpass filter is 1 - Hlp,
% where Hlp is the transfer function of the corresponding lowpass
% filter. Thus, we can use function lpfilter to generate highpass
% filters.
```

```
if nargin == 4
   n = 1; % Default value of n.
end
% Generate highpass filter.
Hlp = lpfilter(type, M, N, D0, n);
H = 1 - Hlp;
```

例 4.6　高通滤波器

图 4.17 显示了理想的巴特沃兹和高斯高通滤波器的图形与图像。图 4.17(a)所示的图形是使用如下命令得到的：

```
>> H = fftshift(hpfilter('ideal', 500, 500, 50));
>> mesh(H(1:10:500, 1:10:500));
>> axis([0 50 0 50 0 1])
>> colormap([0 0 0])
>> axis off
>> grid off
```

图 4.17(d)所示的相应图像是使用命令

```
>>figure, imshow(H, [ ])
```

得到的，其中，添加到图像上的细黑边描绘了其边界。类似的命令生成图 4.17 的其余部分（二阶巴特沃兹滤波器）。

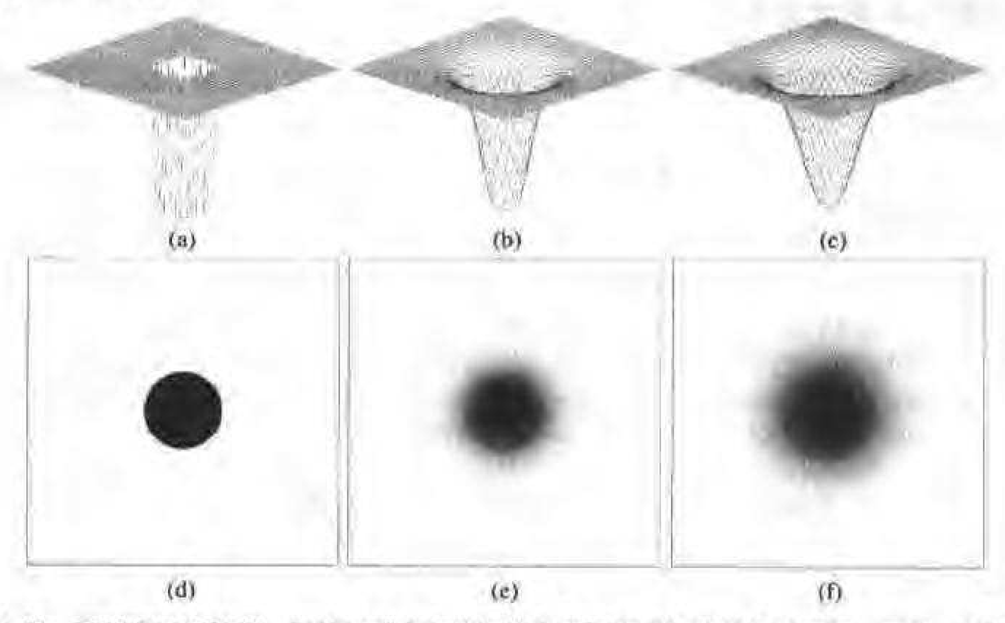


图 4.17　上行：理想高通滤波器、巴特沃兹高通滤波器和高斯高通滤波器的透视图。下行：其相应的图像

例 4.7　高通滤波

图 4.18(a)是与图 4.13(a)相同的测试图案 f。图 4.18(b)是使用下面的命令得到的，它显示了对 f 应用高斯高通滤波器后的结果：

```
>> PQ = paddedsize(size(f));
>> D0 = 0.05*PQ(1);
>> H = hpfilter('gaussian', PQ(1), PQ(2), D0);
>> g = dftfilt(f, H);
>> figure, imshow(g, [ ])
```



图4.18 (a)原图像；(b)高斯高通滤波后的结果

如图4.18(b)所示，图像的边缘和其他亮度急速变化区得到了增强。然而，由于图像的平均值已由$F(0, 0)$给出，而且迄今为止讨论的高通滤波器偏离了傅里叶变换的原点，所以图像失去了大部分原图像所呈现出的背景色调。这个问题将在下一节中处理。

4.6.2 高频强调滤波

正如例4.7提及的那样，高通滤波器偏离了直流项，从而把图像的平均值降低到了零。一种补偿方法是给高通滤波器加上一个偏移量。若偏移量与将滤波器乘以一个大于1的常数结合起来，则这种方法就称为高频强调滤波，因为该常量乘数突出了高频部分。这个乘数也增加了低频部分的幅度，但是只要偏移量与乘数相比较小，低频增强的影响就弱于高频增强的影响。高频强调滤波器的传递函数为

$$H_{\mathrm{hfe}}(u, v) = a + bH_{\mathrm{hp}}(u, v)$$

其中，a是偏移量，b是乘数，$H_{\mathrm{hp}}(u, v)$是高通滤波器的传递函数。

例4.8 将高频强调滤波与直方图均衡化结合起来

图4.19(a)显示了胸部的一幅X光图像f。由于X光图像不能像光学透镜那样聚焦，所以结果图像略显模糊。本例的目的是锐化图4.19(a)。由于这幅特殊图像的灰度偏向于灰度级的暗端，所以我们还将利用这个机会说明如何使用空间域处理来补偿频域滤波。

图4.19(b)显示了用二阶巴特沃兹高通滤波器对图4.19(a)滤波的结果，D_0的值等于已填充图像垂直尺寸的5%。只要滤波器的半径不小到使得变换原点附近的频率通过，高通滤波就不会对D_0的值过度敏感。正如所预料的那样，滤波的结果并无特色，但它模糊地显示出了图像的主要边缘。高频强调滤波（此时$a = 0.5$，$b = 2.0$）的优点显示在图4.19(c)中，图像中由低频成分引起的灰度级色调得以保持。下列命令用以产生图4.19所示的处理后的图像，其中f表示输入图像［最后一条命令产生了图4.19(d)］：

```
>> PQ = paddedsize(size(f));
>> D0 = 0.05*PQ(1);
>> HBW = hpfilter('btw', PQ(1), PQ(2), D0, 2);
>> H = 0.5 + 2*HBW;
>> gbw = dftfilt(f, HBW);
>> gbw = gscale(gbw);
```

```
>> ghf = dftfilt(f, H);
>> ghf = gscale(ghf);
>> ghe = histeq(ghf, 256);
```

正如3.3.2节中指出的那样，在窄范围灰度级中以灰度为特征的图像是直方图均衡化的理想选择。如图4.19(d)所示，这确实是本例中进一步增强图像的合理方法。可以看出，清楚的骨骼结构和其他细节在其他任何三幅图像中是完全看不到的。最终增强的图像有少量噪声，但这是灰度级扩展后典型的X光图像。将高频强调滤波与直方图均衡化结合起来使用时，所得到的结果要好于单独使用任何一种方法所得到的结果。

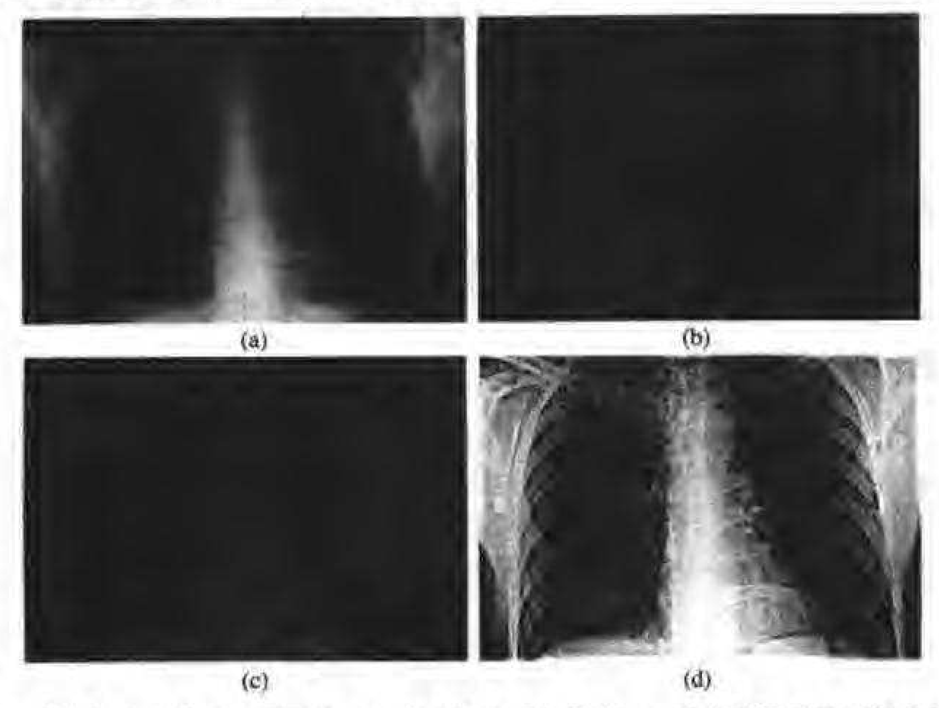


图4.19 高频强调滤波：(a)原图像；(b)高通滤波后的结果；(c)高频强调滤波后的结果；(d)图像(c)经直方图均衡化后的图像（原图像由密歇根州医学院解剖室的Thomas R .Gest提供）

小结

除本章和第3章中用做示例的图像增强应用之外，在这两章中详述的概念和技术为本书后续章节将要讨论的其他图像处理领域奠定了基础。亮度变换常用于亮度缩放，空间域滤波广泛用于第5章中的图像复原、第6章中的彩色处理、第10章中的图像分割以及第11章中的图像描绘子提取。本章详述的傅里叶变换广泛用于第5章中的图像复原、第8章中的图像压缩以及第11章中的图像描述。

第5章 图像复原

前言

复原的目的是在预定义的意义上改善给定的图像。尽管图像增强和图像复原之间有重叠的部分，但前者主要是主观的处理，而图像复原大部分是客观的处理。复原通过使用退化现象的先验知识试图重建或恢复一幅退化的图像。因此，复原技术趋向于将退化模型化并用相反的处理来恢复原图像。

这种方法通常包含把优势准则公式化，它将产生一个希望结果的最优估计。相比而言，为了利用人类视觉系统的心理物理特性，增强技术基本上是以启发式过程来处理一幅图像的。例如，对比度拉伸是受关注的增强技术，因为它主要是给查看者提供一幅赏心悦目的图像，而复原技术则考虑用去模糊函数来消除图像的模糊。

本章主要研究如何使用MATLAB和IPT把退化现象模型化及将复原的解决方案公式化。如在第3章和第4章中，有些复原技术可在空间域最好地公式化，而另一些则更适合在频域公式化。两种方法都将在下面几节中加以研究。

5.1 图像退化/复原处理的模型

如图5.1所示，本章中用退化函数把退化过程模型化，它和加性噪声项一起，作用于输入图像 $f(x, y)$，产生一幅退化的图像 $g(x, y)$：

$$g(x, y) = H[f(x, y)] + \eta(x, y)$$

给定 $g(x, y)$、一些关于退化函数 H 的知识以及一些关于加性噪声 $\eta(x, y)$ 的知识，复原的目标就是得到原图像的一个估计 $\hat{f}(x, y)$。我们要使这个估计尽可能地接近原始的输入图像。通常，我们对 H 和 η 知道得越多，$\hat{f}(x, y)$ 就越接近 $f(x, y)$。

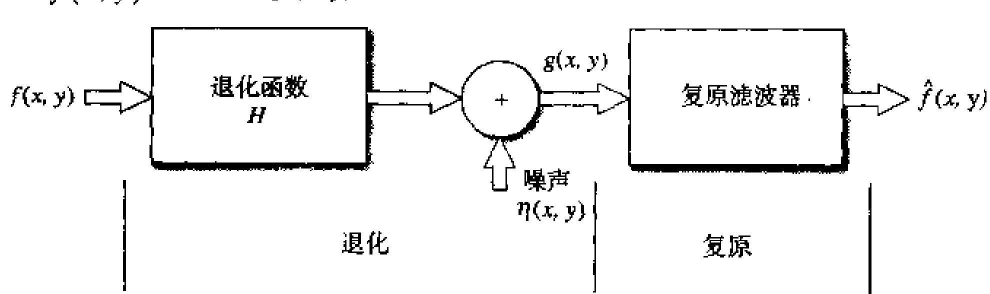


图5.1 图像退化/复原处理的模型

若 H 是线性的、空间不变的过程，则退化图像在空间域通过下式给出：

$$g(x, y) = h(x, y) * f(x, y) + \eta(x, y)$$

其中，$h(x, y)$是退化函数的空间表示，如第4章那样，符号“*”表示卷积。由4.3.1节的讨论可知，空间域的卷积和频域的乘法组成了一个傅里叶变换对，所以可以用等价的频域表示来写出前面的模型：

$$G(u, v) = H(u, v)F(u, v) + N(u, v)$$

其中，用大写字母表示的项是卷积方程式中相应项的傅里叶变换。退化函数 $H(u, v)$ 有时称为光学传递函数（OTF），该名词来源于光学系统的傅里叶分析。在空间域，$h(x, y)$ 称为点扩散函数（PSF）。对于任何种类的输入，让 $h(x, y)$ 作用于光源的一个点来得到退化的特征，点扩散函数就是来源于此的一个名词。OTF和PSF是一个傅里叶变换对，工具箱提供了两个函数 otf2psf 和 psf2otf，用于OTF和PSF之间的转换。

由于退化是线性的，所以空间不变的退化函数 H 可以被模型化为卷积，有时退化过程称为"用PSF或OTF对图像进行卷积"。同样地，复原处理有时也称为反卷积。

在下面的三节中，我们假设 H 是恒等的算子，仅处理由噪声造成的退化。5.6节中将开始考虑几种 H 和 η 都出现的图像复原方法。

5.2　噪声模型

模拟噪声的行为和影响的能力是图像复原的核心。在本章中，我们对两种基本噪声模型感兴趣：空间域的噪声（用噪声概率密度函数来描述）和频域的噪声（用噪声的各种傅里叶特性来描述）。除了5.2.3节中的资料外，我们将假设本章的噪声与图像的坐标无关。

5.2.1　使用函数 imnoise 添加噪声

工具箱采用函数 imnoise 来使用噪声污染一幅图像。该函数的基本语法为

```
g = imnoise(f, type, parameters)
```

其中，f 是输入图像，type 和 parameters 将在后面解释。函数 imnoise 在给图像添加噪声之前，将它转换为范围[0, 1]内的 double 类图像。指定噪声参数时必须考虑到这一点。例如，要将均值为64、方差为400的高斯噪声添加到一幅 uint8 类图像上，我们可将均值标度为64/255，将方差标度为 $400/(255)^2$，以便作为函数 imnoise 的输入。该函数的语法形式如下所示：

- g = imnoise(f,'gaussian',m,var) 将均值为m、方差为var的高斯噪声加到图像f上。默认值为均值是0、方差是0.01的噪声。
- g = imnoise(f,'localvar',V) 将均值为0、局部方差为V的高斯噪声添加到图像f上，其中V是与f大小相同的一个数组，它包含了每个点的理想方差值。
- g = imnoise(f,'localvar,image_intensity,var) 将均值为0的高斯噪声添加到图像f上，其中噪声的局部方差 var 是图像 f 的亮度值的函数。参量 image_intensity 和 var是大小相同的向量，plot(image_intensity, var) 绘制出噪声方差和图像亮度的函数关系。向量 image_intensity 必须包含范围在[0, 1]内的归一化亮度值。
- g = imnoise(f,'salt & pepper',d) 用椒盐噪声污染图像 f，其中 d 是噪声密度（即包含噪声值的图像区域的百分比）。因此，大约有 d*numel(f) 个像素受到了影响。默认的噪声密度为0.05。
- g = imnoise(f,'speckle',var) 用方程 g = f + n*f 将乘性噪声添加到图像 f 上，其中 n 是均值为0、方差为 var 的均匀分布的随机噪声。var 的默认值为0.04。
- g = imnoise(f,'poisson') 从数据中生成泊松噪声，而不是将人工的噪声添加到数据中。为了遵守泊松统计，uint8 类和 uint16 类图像的亮度必须和光子（或其他任何的量子信息）的数量相符合。当每像素的光子数量大于65 535（但小于 10^{12}）时，就要使用双精度图像。亮度值在0和1之间变化，并且对应于光子的数量除以 10^{12}。

函数 imnoise 的一些说明将在下面几节中给出。

表 5.1 随机变量的生成

名称	PDF	均值和方差	CDF	生成器*
均匀	$p_z(z)=\begin{cases}\frac{1}{b-a} & 若 a\leqslant z\leqslant b\\ 0 & 其他\end{cases}$	$m=\frac{a+b}{2},\quad \sigma^2=\frac{(b-a)^2}{12}$	$F_z(z)=\begin{cases}0 & z<a\\ \frac{z-a}{b-a} & a\leqslant z\leqslant b\\ 1 & z>b\end{cases}$	MATLAB函数rand
高斯	$p_z(z)=\frac{1}{\sqrt{2\pi}b}e^{-(z-a)^2/2b^2}$ $-\infty<z<\infty$	$m=a,\quad \sigma^2=b^2$	$F_z(z)=\int_{-\infty}^{z}p_z(v)\,dv$	MATLAB函数randn
椒盐	$p_z(z)=\begin{cases}P_a & z=a\\ P_b & z=b\\ 0 & 其他\end{cases}$ $b>a$	$m=aP_a+bP_b$ $\sigma^2=(a-m)^2P_a+(b-m)^2P_b$	$F_z(z)=\begin{cases}0 & 对于\ z<a\\ P_a & 对于\ a\leqslant z<b\\ P_a+P_b & 对于\ b\leqslant z\end{cases}$	具有附加逻辑的MATLAB函数rand
对数正态	$p_z(z)=\frac{1}{\sqrt{2\pi}bz}e^{-[\ln(z)-a]^2/2b^2}$ $z>0$	$m=e^{a+(b^2/2)},\sigma^2=[e^{b^2}-1]e^{2a+b^2}$	$F_z(z)=\int_0^z p_z(v)\,dv$	$z=ae^{bN(0,1)}$
瑞利	$p_z(z)=\begin{cases}\frac{2}{b}(z-a)e^{-(z-a)^2/b} & z\geqslant a\\ 0 & z<a\end{cases}$	$m=a+\sqrt{\pi b/4},\sigma^2=\frac{b(4-\pi)}{4}$	$F_z(z)=\begin{cases}1-e^{-(z-a)^2/b} & z\geqslant a\\ 0 & z<a\end{cases}$	$z=a+\sqrt{b\ln[1-U(0,1)]}$
指数	$p_z(z)=\begin{cases}ae^{-az} & z\geqslant 0\\ 0 & z<0\end{cases}$	$m=\frac{1}{a},\quad \sigma^2=\frac{1}{a^2}$	$F_z(z)=\begin{cases}1-e^{-az} & z\geqslant 0\\ 0 & z<0\end{cases}$	$z=-\frac{1}{a}\ln[1-U(0,1)]$
厄兰	$p_z(z)=\frac{a^b z^{b-1}}{(b-1)!}e^{-az}$ $z\geqslant 0$	$m=\frac{b}{a},\quad \sigma^2=\frac{b}{a^2}$	$F_z(z)=\left[1-e^{-az}\sum_{n=0}^{b-1}\frac{(az)^n}{n!}\right]$ $z\geqslant 0$	$z=E_1+E_2+\cdots+E_b$ E_i是具有参数a的指数随机数

*$N(0, 1)$表示正态（高斯）随机数，其均值为0，方差为1。$U(0, 1)$表示范围$(0, 1)$内的均匀随机数。

与imnoise不同，下面的M函数产生一个大小为M × N的噪声数组R，它不以任何方式缩放。另一个主要的不同是imnoise输出一个有噪声的图像，而imnoise2产生噪声模式本身。用户可直接指定需要的噪声参数。注意，由椒盐噪声产生的噪声数组有三个值：相应于胡椒噪声的0，相应于盐粒噪声的1，以及相应于无噪声的0.5。

为了使这个数组有用，需要对它进行进一步的处理。例如，若要用这个数组来污染一幅图像，则我们可使用函数find寻找R中所有值为0的坐标，并把图像中相应的坐标置为可能的最小灰度值（通常是0）。同样，我们可寻找R中所有值为1的坐标，并把图像中相应的坐标值置为可能的最高值（对于8比特图像来说通常是255）。该处理模拟了椒盐噪声实际上是怎样影响一幅图像的。

```
function R = imnoise2(type, M, N, a, b)
%IMNOISE2 Generates an array of random numbers with specified PDF.
%   R = IMNOISE2(TYPE, M, N, A, B) generates an array, R, of size
%   M-by-N, whose elements are random numbers of the specified TYPE
%   with parameters A and B. If only TYPE is included in the
%   input argument list, a single random number of the specified
%   TYPE and default parameters shown below is generated. If only
%   TYPE, M, and N are provided, the default parameters shown below
%   are used. If M = N = 1, IMNOISE2 generates a single random
%   number of the specified TYPE and parameters A and B.
%
%   Valid values for TYPE and parameters A and B are:
%
%   'uniform'         Uniform random numbers in the interval (A, B).
%                     The default values are (0, 1).
%   'gaussian'        Gaussian random numbers with mean A and standard
%                     deviation B. The default values are A = 0, B = 1.
%   'salt & pepper'   Salt and pepper numbers of amplitude 0 with
%                     probability Pa = A, and amplitude 1 with
%                     probability Pb = B. The default values are Pa =
%                     Pb = A = B = 0.05. Note that the noise has
%                     values 0 (with probability Pa = A) and 1 (with
%                     probability Pb = B), so scaling is necessary if
%                     values other than 0 and 1 are required. The noise
%                     matrix R is assigned three values. If R(x, y) =
%                     0, the noise at (x, y) is pepper (black). If
%                     R(x, y) = 1, the noise at (x, y) is salt
%                     (white). If R(x, y) = 0.5, there is no noise
%                     assigned to coordinates (x, y).
%   'lognormal'       Lognormal numbers with offset A and shape
%                     parameter B. The defaults are A = 1 and B =
%                     0.25.
%   'rayleigh'        Rayleigh noise with parameters A and B. The
%                     default values are A = 0 and B = 1.
%   'exponential'     Exponential random numbers with parameter A. The
%                     default is A = 1.
%   'erlang'          Erlang (gamma) random numbers with parameters A
%                     and B. B must be a positive integer. The
%                     defaults are A = 2 and B = 5. Erlang random
%                     numbers are approximated as the sum of B
%                     exponential random numbers.
% Set default values.
```

```
if nargin == 1
   a = 0; b = 1;
   M = 1; N = 1;
elseif nargin == 3
   a = 0; b = 1;
end

% Begin processing. Use lower(type) to protect against input
% being capitalized.
switch lower(type)
case 'uniform'
   R = a + (b - a)*rand(M, N);
case 'gaussian'
   R = a + b*randn(M, N);
case 'salt & pepper'
   if nargin <= 3
      a = 0.05; b = 0.05;
   end
   % Check to make sure that Pa + Pb is not > 1.
   if (a + b) > 1
      error('The sum Pa + Pb must not exceed 1.')
   end
   R(1:M, 1:N) = 0.5;
   % Generate an M-by-N array of uniformly-distributed random numbers
   % in the range (0, 1). Then, Pa*(M*N) of them will have values <=
   % a. The coordinates of these points we call 0 (pepper
   % noise). Similarly, Pb*(M*N) points will have values in the range
   % > a & <= (a + b). These we call 1 (salt noise).
   X = rand(M, N);
   c = find(X <= a);
   R(c) = 0;
   u = a + b;
   c = find(X > a & X <= u);
   R(c) = 1;
case 'lognormal'
   if nargin <= 3
      a = 1; b = 0.25;
   end
   R = a*exp(b*randn(M, N));
case 'rayleigh'
   R = a + (-b*log(1 - rand(M, N))).^0.5;
case 'exponential'
   if nargin <= 3
      a = 1;
   end
   if a <= 0
      error('Parameter a must be positive for exponential type.')
   end
   k = -1/a;
   R = k*log(1 - rand(M, N));
case 'erlang'
   if nargin <= 3
      a = 2; b = 5;
   end
   if (b ~= round(b) | b <= 0)
```

```
            error('Param b must be a positive integer for Erlang.')
         end
         k = -1/a;
         R = zeros(M, N);
         for j = 1:b
            R = R + k*log(1 - rand(M, N));
         end
      otherwise
         error('Unknown distribution type.')
      end
```

例 5.2 使用函数 imnoise2 产生数据的直方图

图5.2显示了表5.1中所示的所有类型的随机数的直方图。每个图形的数据都是用函数imnoise2生成的。例如，图 5.2(a)的数据是用下面的命令产生的：

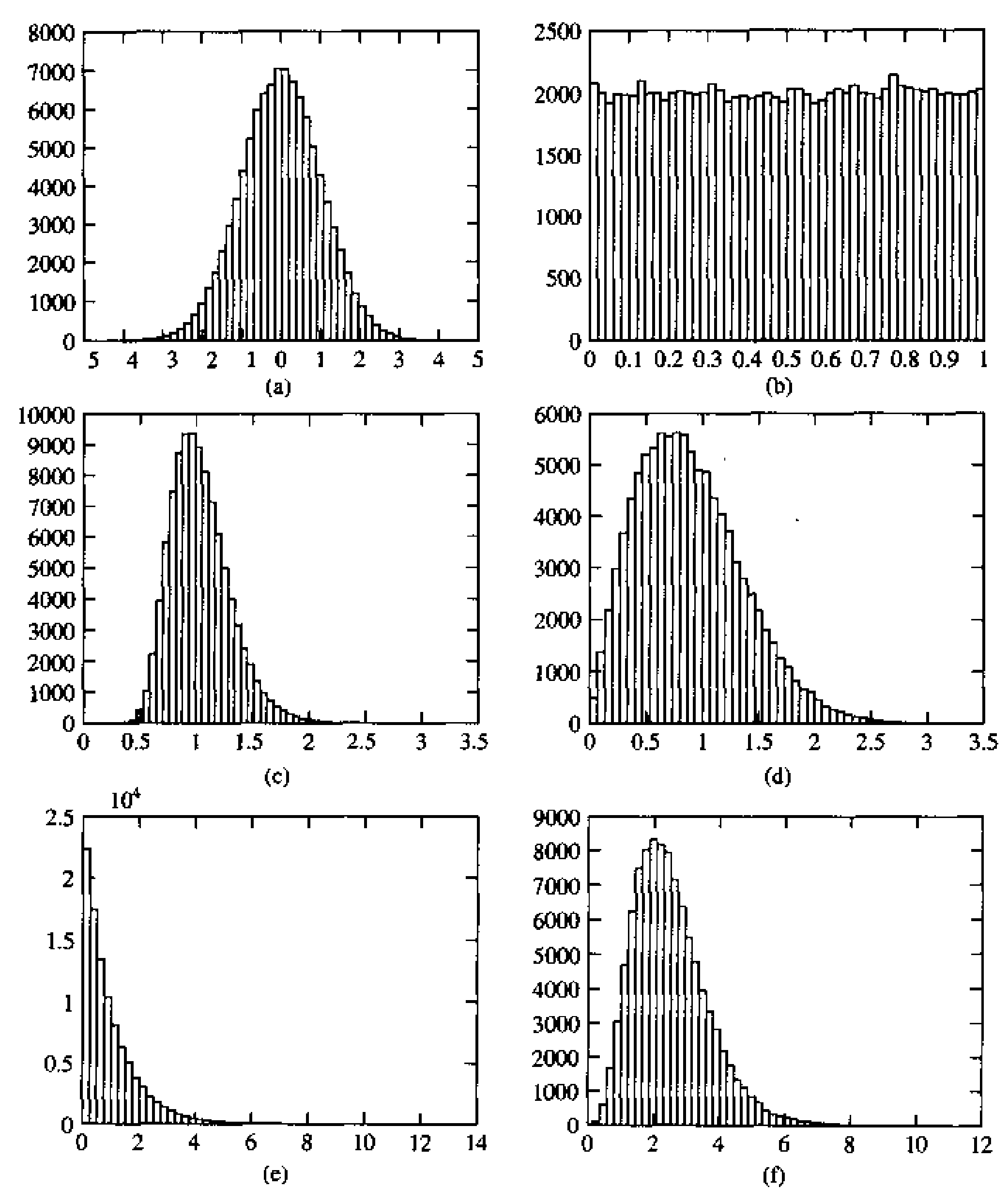


图 5.2 随机数的直方图：(a)高斯；(b)均匀；(c)对数正态；(d)瑞利；(e)指数；(f)厄兰。每种情况都使用了在函数 imnoise2 的说明中列出的默认参数

```
>> r = imnoise2('gaussian', 100000, 1, 0, 1);
```

这条语句产生一个有着100 000个元素的列向量r，每个元素都来自均值为0、标准偏差为1的高斯分布的随机数。直方图可以使用函数hist得到，函数hist的语法为

```
p = hist(r, bins)
```

其中，bins是bin的数目。我们用bins = 50来产生图5.2所示的直方图。其他直方图也可通过类似的方式得到。对每种情况，所选的参数都是在函数imnoise2的说明里列出的默认值。

5.2.3 周期噪声

图像的周期噪声一般产生于图像采集过程中的电气和/或电机干扰。这是本章惟一考虑的一种空间从属噪声。如5.4节讨论的那样，图像的周期噪声通常是通过频域滤波来处理的。我们的周期噪声的模型是二维正弦波，它有如下所示的方程：

$$r(x, y) = A\sin[2\pi u_0(x + B_x)/M + 2\pi v_0(y + B_y)/N]$$

其中，A是振幅，u_0和v_0分别关于x轴和y轴确定正弦频率。B_x和B_y是关于原点的相移。该方程的$M \times N$ DFT为

$$R(u, v) = j\frac{A}{2}[(e^{j2\pi u_0 B_x/M})\delta(u + u_0, v + v_0) - (e^{j2\pi v_0 B_y/N})\delta(u - u_0, v - v_0)]$$

我们所看到的是一对分别位于$(u + u_0, v + v_0)$和$(u - u_0, v - v_0)$的复共轭冲击。

下面的M函数接受冲击位置（频率坐标）的一个任意数，每个冲击位置都有自己的振幅、频率和相移参数，并且以前一段中所描述的正弦和的形式计算$r(x, y)$。这个函数也输出正弦波之和的傅里叶变换$R(u, v)$以及$R(u, v)$的谱。正弦波通过逆DFT从给定的冲击位置信息中产生。这使得它更加直观，而且也简化了空间噪声模式中的频率显示。确定一个冲击的位置仅需要一对坐标。这个程序将产生共轭对称冲击（注意，为了像在4.2节中那样进行ifft2操作，代码中使用了函数ifftshift，以便将居中的R转换为合适的数据排列）。

```
function [r, R, S] = imnoise3(M, N, C, A, B)
%IMNOISE3 Generates periodic noise.
%   [r, R, S] = IMNOISE3(M, N, C, A, B), generates a spatial
%   sinusoidal noise pattern, r, of size M-by-N, its Fourier
%   transform, R, and spectrum, S. The remaining parameters are as
%   follows:
%
%   C is a K-by-2 matrix containing K pairs of frequency domain
%   coordinates (u, v) indicating the locations of impulses in the
%   frequency domain. These locations are with respect to the
%   frequency rectangle center at (M/2 + 1, N/2 + 1). Only one pair
%   of coordinates is required for each impulse. The program
%   automatically generates the locations of the conjugate symmetric
%   impulses. These impulse pairs determine the frequency content
%   of r.
%
%   A is a 1-by-K veotor that contains the amplitude of each of the
%   K impulse pairs. If A is not included in the argument, the
```

```
%   default used is A = ONES(1, K). B is then automatically set to
%   its default values (see next paragraph). The value specified
%   for A(j) is associated with the coordinates in C(j, 1:2).
%
%   B is a K-by-2 matrix containing the Bx and By phase components
%   for each impulse pair. The default values for B are B(1:K, 1:2)
%   = 0.

% Process input parameters.
[K, n] = size(C);
if nargin == 3
   A(1:K) = 1.0;
   B(1:K, 1:2) = 0;
elseif nargin == 4
   B(1:K, 1:2) = 0;
end

% Generate R.
R = zeros(M, N);
for j = 1:K
   u1 = M/2 + 1 + C(j, 1); v1 = N/2 + 1 + C(j, 2);
   R(u1, v1) = i * (A(j)/2) * exp(i*2*pi*C(j, 1) * B(j, 1)/M);
   % Complex conjugate.
   u2 = M/2 + 1 - C(j, 1); v2 = N/2 + 1 - C(j, 2);
   R(u2, v2) = -i * (A(j)/2) * exp(i*2*pi*C(j, 2) * B(j, 2)/N);
end

% Compute spectrum and spatial sinusoidal pattern.
S = abs(R);
r = real(ifft2(ifftshift(R)));
```

例5.3 使用函数 imnoise3

图5.3(a)和图5.3(b)显示了使用下面的命令产生的频谱和空间正弦噪声模式：

```
>> C = [0 64; 0 128; 32 32; 64 0; 128 0; -32 32];
>> [r, R, S] = imnoise3(512, 512, C);
>> imshow(S, [ ])
>> figure, imshow(r, [ ])
```

回顾可知坐标的次序是(u, v)。这两个值是参照频率矩形的中心指定的（见4.2节中关于这个中心点的坐标的定义）。图5.3(c)和图5.3(d)显示了重复前面的命令得到的结果，但其中

```
>> C = [0 32; 0 64; 16 16; 32 0; 64 0; -16 16];
```

同样，图5.3(e)是通过

```
>> C = [6 32; -2 2];
```

得到的。

图5.3(f)是用同样的C产生的，但是使用了非默认的振幅向量：

```
>> A = [1 5];
>> [r, R, S] = imnoise3(512, 512, C, A);
```

如图5.3(f)所示，较低频率的正弦波支配了图像。这正如所预料的那样，因为它的振幅是较高频率分量振幅的五倍。

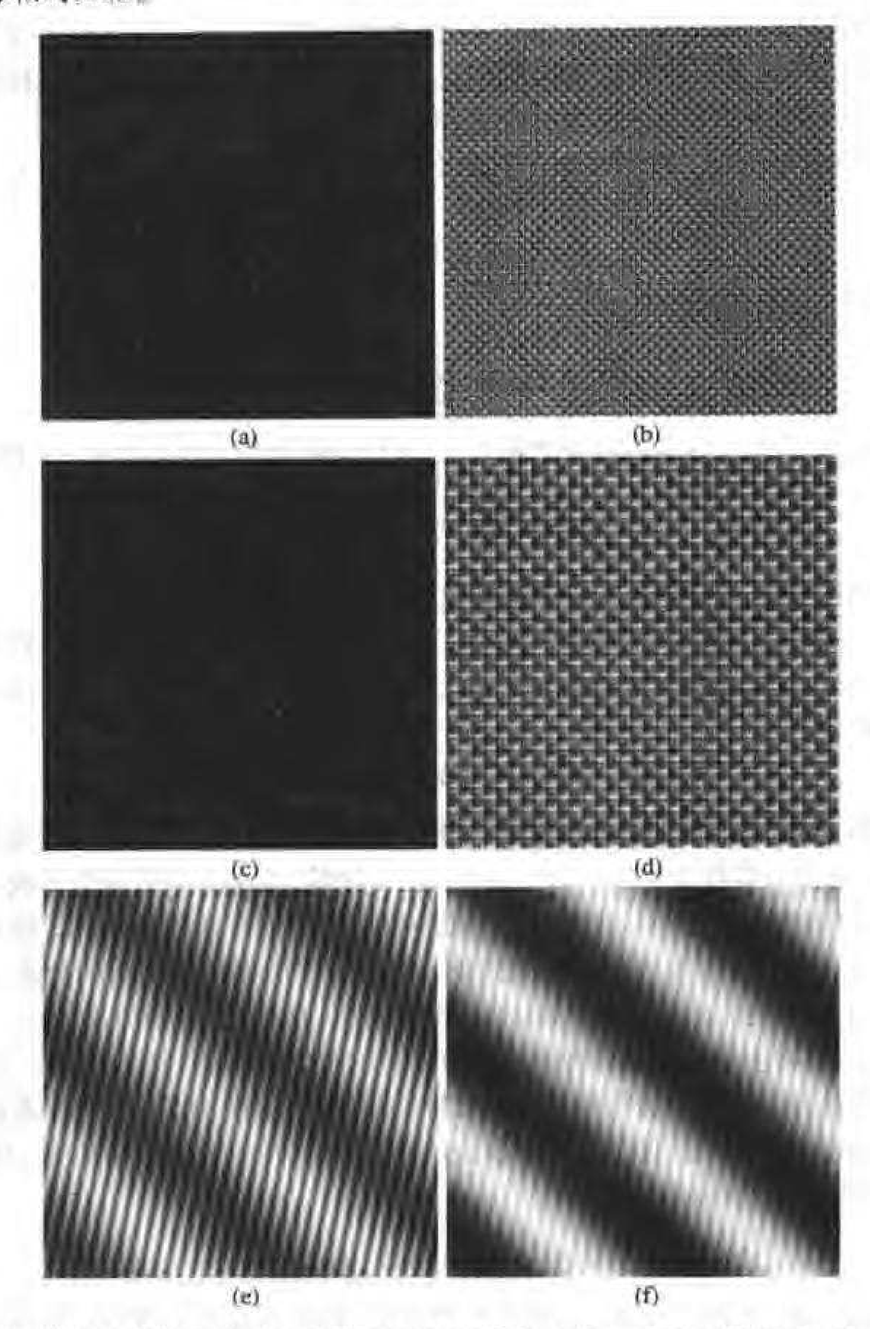


图5.3 (a)指定冲击的频谱；(b)相应的正弦噪声模式；(c)和(d)相似的序列；(e)和(f)两种其他的噪声模式。为了使(a)和(c)中的点更容易看到，已对它们进行了放大

5.2.4 估计噪声参数

周期噪声的参数一般是通过分析图像的傅里叶频谱来估计的。周期噪声往往产生频率尖峰，该尖峰常常可以通过目视来检测。在噪声尖峰明显或知道了干扰频率的一些知识的情况下，自动分析就是有可能的。

在空间域噪声的情况下，PDF的参数可能通过传感器的技术参数部分地知道，但是通常通过样本图像来估计它们是很必要的。噪声的均值m和方差σ^2的关系，以及用来指定本章中的噪声PDF

的参数a和b，都已在表5.1中列出。因此，问题就变成了通过样本图像来估计均值和方差，然后利用这些估计来求解a和b。

假设z_i是用来表示一幅图像的灰度级的一个离散随机变量，令$p(z_i), i = 0, 1, 2, \cdots, L-1$是相应的归一化直方图，其中$L$是可能的亮度值的数目。直方图分量$p(z_i)$是亮度值$z_i$出现的概率的一个估计，该直方图也可以看成是亮度PDF的一个近似。

描述直方图分布形状的一种主要方法是通过它的中心矩（也称为均值的矩），它定义为

$$\mu_n = \sum_{i=0}^{L-1}(z_i - m)^n p(z_i)$$

其中，n是矩的阶，m是均值：

$$m = \sum_{i=0}^{L-1} z_i p(z_i)$$

因为假设直方图已归一化，其所有分量之和为1，所以由前面的方程可知$\mu_0 = 1$和$\mu_1 = 0$。二阶矩

$$\mu_2 = \sum_{i=0}^{L-1}(z_i - m)^2 p(z_i)$$

是方差。在本章中，我们仅对均值和方差感兴趣。高阶矩将在第11章中讨论。

函数statmoments计算均值和n阶中心矩，并返回行向量v。因为0阶矩总为1，1阶矩总为0，所以statmoments忽略这两个矩，改为令v(1) = m，v(k) = μ_k，$k = 2, 3, \cdots, n$。语法如下所示（代码参看附录C）：

```
[v, unv] = statmoments(p, n)
```

其中，p为直方图向量，n是计算的矩的数量。要求对于uint8类图像，p的分量数等于2^8；对于uint16类图像，p的分量数等于2^{16}；对于double类图像，p的分量数等于2^8或2^{16}。输出向量v包含了以随机变量值为基础的归一化矩，而随机变量已被标度在区间[0, 1]内，所以，所有的矩也在这个区间内。向量unv包含了与v相同的矩，但用位于原始值区间内的数值计算。例如，若length(p) = 256，v(1) = 0.5，则unv(1)的值将为127.5，是区间[0, 255]的一半。

噪声参数通常直接由带噪声的图像或一组带噪声的图像来估计。在这种情况下，所选择的方法是，在一幅图像中选择一个尽可能与背景一样无特色的区域，以便使该区域的亮度值的可变性主要由噪声产生。在MATLAB中，我们使用函数roipoly来选择一个感兴趣区域（ROI），该函数将产生一个多边形的ROI。该函数的语法为

```
B = roipoly(f, c, r)
```

其中，f是我们感兴趣的图像，c和r是多边形顶点的相应列坐标和行坐标（注意，列先被指定）。输出B是一个二值图像，它是一个与f大小相同且在感兴趣的区域外为0、在感兴趣的区域内为1的二值图像。图像B是一个用来将操作限制在感兴趣区域内的掩模。

为了交互式地指定一个多边形的ROI，可使用下面的语法：

```
B = roipoly(f)
```

它将把图像f显示到屏幕上，让用户使用鼠标来指定多边形。若省略了f，则函数roipoly将在最后显示的图像上操作。使用普通的按钮可以增加多边形的顶点。按**Backspace**键或**Delete**键可删除先前选中的顶点。**Shift**键加上单击、右击或双击鼠标可为选区添加最后一个顶点，然后开始用1来填充多边形区域。按**Return**键则会结束选区而不添加任何顶点。

我们，由B定义的区域的可变性主要是由噪声的变化所造成的。若可行，估计噪声的均值和方差的另一种方法是做一个已知灰度的常量对象。图5.4(d)显示了使用一组npix命令（这个数是histroi返回的）得到的高斯随机变量的直方图，这些变量的均值是147，方差是400：

```
>> X = imnoise2('gaussian', npix, 1, 147, 20);
>> figure, hist(X, 130)
>> axis([0 300 0 140])
```

其中，给出hist中的bin数目是为了使结果与图5.4(c)所示的图形一致。这幅图中的直方图是在函数histroi内使用imhist得到的（参看前面的代码），imhist使用了与hist不同的标度。我们选择一组npix随机变量来产生X，以便使两种直方图中的样本数量相同。图5.4(c)和图5.4(d)的相似性清楚地说明了使用带有接近于估计参数v(1)和v(2)的高斯分布，确实有非常好的近似。

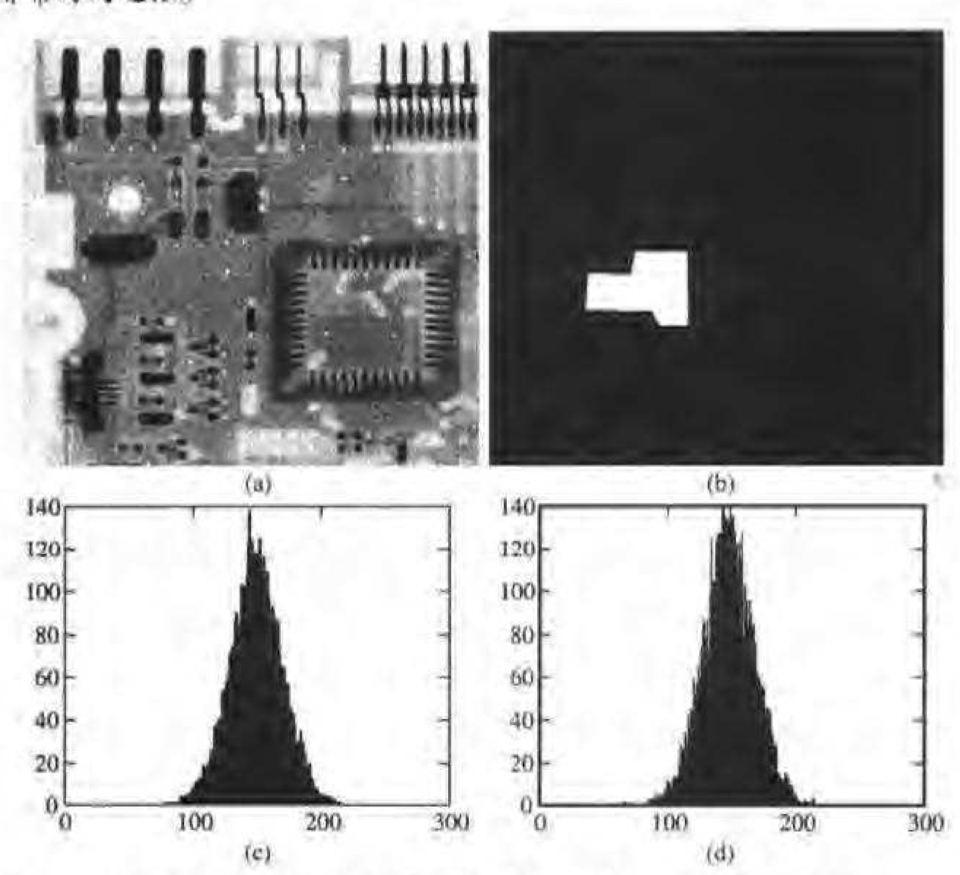


图5.4　(a)噪声图像；(b)交互式地产生的ROI；(c)ROI的直方图；(d)使用函数imnoise2产生的高斯数据的直方图（原图像由Lixi公司提供）

5.3　仅有噪声的复原：空间滤波

若出现的退化仅仅是噪声，则它就遵循5.1节中给出的模型：

$$g(x, y) = f(x, y) + \eta(x, y)$$

在这种情况下，所选择的降低噪声的方法是空间滤波，这里我们使用在3.4节和3.5节中讨论过的相似技术。本节中，我们将总结和实现几种降低噪声的空间滤波器。这些滤波器的其他具体特性见Gonzalez and Woods[2002]。

5.3.1 空间噪声滤波器

表5.2列出了本节所感兴趣的空间滤波器，其中S_{xy}表示输入噪声图像g的$m \times n$子图像（区域）。S的下标表示子图像中心的坐标(x, y)，$\hat{f}(x, y)$（f的一个估值）表示滤波器在这些坐标处的响应。线性滤波器使用3.4节中讨论过的函数imfilter来实现。中值滤波器、最大滤波器和最小滤波器是非线性排序统计滤波器。中值滤波器可以直接使用IPT函数medfilt2来实现。最大和最小滤波器将使用3.5.2节中讨论过的更为通用的排序滤波器函数ordfilt2来实现。

下面我们称为spfilt的函数与表5.2中列出的任何滤波器在空间域执行滤波。注意，（在表2.5中提到的）函数imlincomb的用途是计算输入的线性组合。该函数的语法为

```
B = imlincomb(c1, A1, c2, A2, · · ·, ck, Ak)
```

它实现了等式

```
B = c1*A1 + c2*A2 + · · · + ck*Ak
```

其中，c是double类实标量，A是有着相同类和大小的数组。还要注意在子函数gmean中函数warning是如何被打开和关闭的。在这种情况下，若函数log的参数变为0，则我们会禁止由MATLAB发出的警告。通常，任何程序中都可以使用函数warning。基本语法为

```
warning('message')
```

除了可以使用命令warning on和warning off来打开和关闭外，该函数与函数disp极为相似。

```
function f = spfilt(g, type, m, n, parameter)
%SPFILT Performs linear and nonlinear spatial filtering.
%   F = SPFILT(G, TYPE, M, N, PARAMETER) performs spatial filtering
%   of image G using a TYPE filter of size M-by-N. Valid calls to
%   SPFILT are as follows:
%
%      F = SPFILT(G, 'amean', M, N)       Arithmetic mean filtering.
%      F = SPFILT(G, 'gmean', M, N)       Geometric mean filtering.
%      F = SPFILT(G, 'hmean', M, N)       Harmonic mean filtering.
%      F = SPFILT(G, 'chmean', M, N, Q)   Contraharmonic mean
%                                         filtering of order Q. The
%                                         default is Q = 1.5.
%      F = SPFILT(G, 'median', M, N)      Median filtering.
%      F = SPFILT(G, 'max', M, N)         Max filtering.
%      F = SPFILT(G, 'min', M, N)         Min filtering.
%      F = SPFILT(G, 'midpoint', M, N)    Midpoint filtering.
%      F = SPFILT(G, 'atrimmed', M, N, D) Alpha-trimmed mean filtering.
%                                         Parameter D must be a nonnegative even
%                                         integer; its default value
%                                         is D = 2.
%
%   The default values when only G and TYPE are input are M = N = 3,
%   Q = 1.5, and D = 2.

% Process inputs.
if nargin == 2
   m = 3; n = 3; Q = 1.5; d = 2;
elseif nargin == 5
```

```
    Q = parameter; d = parameter;
elseif nargin == 4
    Q = 1.5; d = 2;
else
    error('Wrong number of inputs.');
end

% Do the filtering.
switch type
case 'amean'
    w = fspecial('average', [m n]);
    f = imfilter(g, w, 'replicate');
case 'gmean'
    f = gmean(g, m, n);
case 'hmean'
    f = harmean(g, m, n);
case 'chmean'
    f = charmean(g, m, n, Q);
case 'median'
    f = medfilt2(g, [m n], 'symmetric');
case 'max'
    f = ordfilt2(g, m*n, ones(m, n), 'symmetric');
case 'min'
    f = ordfilt2(g, 1, ones(m, n), 'symmetric');
case 'midpoint'
    f1 = ordfilt2(g, 1, ones(m, n), 'symmetric');
    f2 = ordfilt2(g, m*n, ones(m, n), 'symmetric');
    f = imlincomb(0.5, f1, 0.5, f2);
case 'atrimmed'
    if (d < 0) | (d/2 ~= round(d/2))
        error('d must be a nonnegative, even integer.')
    end
    f = alphatrim(g, m, n, d);
otherwise
    error('Unknown filter type.')
end

%-------------------------------------------------------------------%
function f = gmean(g, m, n)
%  Implements a geometric mean filter.
inclass = class(g);
g = im2double(g);
% Disable log(0) warning.
warning off;
f = exp(imfilter(log(g), ones(m, n), 'replicate')) . ^ (1 / m / n);
warning on;
f = changeclass(inclass, f);

%-------------------------------------------------------------------%
function f = harmean(g, m, n)
%  Implements a harmonic mean filter.
inclass = class(g);
g = im2double(g);
f = m * n ./ imfilter(1./(g + eps), ones(m, n), 'replicate');
```

```
f = changeclass(inclass, f);

%------------------------------------------------------------------%
function f = charmean(g, m, n, q)
%  Implements a contraharmonic mean filter.
inclass = class(g);
g = im2double(g);
f = imfilter(g.^(q+1), ones(m, n), 'replicate');
f = f ./ (imfilter(g.^q, ones(m, n), 'replicate') + eps);
f = changeclass(inclass, f);

%------------------------------------------------------------------%
function f = alphatrim(g, m, n, d)
%  Implements an alpha-trimmed mean filter.
inclass = class(g);
g = im2double(g);
f = imfilter(g, ones(m, n), 'symmetric');
for k = 1:d/2
   f = imsubtract(f, ordfilt2(g, k, ones(m, n), 'symmetric'));
end
for k = (m*n - (d/2) + 1):m*n
   f = imsubtract(f, ordfilt2(g, k, ones(m, n), 'symmetric'));
end
f = f / (m*n - d);
f = changeclass(inclass, f);
```

例5.5 使用函数 spfilt

图5.5(a)所示的图像是一幅被概率只有0.1的胡椒噪声污染的uint8类图像。这幅图像是使用下面的命令产生的［f表示3.18(a)所示的原图像］：

```
>> [M, N] = size(f);
>> R = imnoise2('salt & pepper', M, N, 0.1, 0);
>> c = find(R == 0);
>> gp = f;
>> gp(c) = 0;
```

图5.5(b)所示的图像仅被盐粒噪声污染，它是使用下面的语句产生的：

```
>> R = imnoise2('salt & pepper', M, N, 0, 0.1);
>> c = find(R == 1);
>> gs = f;
>> gs(c) = 255;
```

过滤胡椒噪声的较好办法是使用Q为正值的反调和滤波器。图5.5(c)是使用下面的语句产生的：

```
>> fp = spfilt(gp, 'chmean', 3, 3, 1.5);
```

同样，盐粒噪声可以使用Q为负值的反调和滤波器滤波：

```
>> fs = spfilt(gs, 'chmean', 3, 3, -1.5);
```

图5.5(d)显示了结果。使用最大和最小滤波器可以得到类似的结果。例如，图5.5(e)和图5.5(f)所示的图像分别是对图5.5(a)和图5.5(b)使用下面的命令产生的：

```
>> fpmax = spfilt(gp, 'max', 3, 3);
>> fsmin = spfilt(gs, 'min', 3, 3);
```

使用spfilt的其他方法可通过类似的方式实现。

表 5.2 空间滤波器。变量 m 和 n 分别表示滤波器邻域的行数和列数

滤波器名称	公式	说明
算术平均	$\hat{f}(x,y)=\frac{1}{mn}\sum_{(s,t)\in S_{xy}} g(s,t)$	用IPT函数w = fspecial('average',[m,n])和f = imfilter(g,w)实现
几何平均	$\hat{f}(x,y)=\left[\prod_{(s,t)\in S_{xy}} g(s,t)\right]^{\frac{1}{mn}}$	该非线性滤波器用函数gmean实现（见本节中的常规函数spfilt）
调和均值	$\hat{f}(x,y)=\frac{mn}{\sum_{(s,t)\in S_{xy}}\frac{1}{g(s,t)}}$	该非线性滤波器用函数harmean实现（见本节中的常规函数spfilt）
反调和均值	$\hat{f}(x,y)=\frac{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q+1}}{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q}}$	该非线性滤波器用函数charmean实现（见本节常规函数spfilt）
中值	$\hat{f}(x,y)=\underset{(s,t)\in S_{xy}}{\text{median}}\{g(s,t)\}$	用IPT函数medfilt2实现：f = medfilt2(g,[m n])
最大	$\hat{f}(x,y)=\max_{(s,t)\in S_{xy}}\{g(s,t)\}$	用IPT函数ordfilt2实现：f = ordfilt2(g,m*n,ones(m,n))
最小	$\hat{f}(x,y)=\min_{(s,t)\in S_{xy}}\{g(s,t)\}$	用IPT函数ordfilt2实现：f = ordfilt2(g,1,ones(m,n))
中点	$\hat{f}(x,y)=\frac{1}{2}\left[\max_{(s,t)\in S_{xy}}\{g(s,t)\}+\min_{(s,t)\in S_{xy}}\{g(s,t)\}\right]$	以最大最小滤波操作的0.5倍实现
顺序-平衡均值	$\hat{f}(x,y)=\frac{1}{mn-d}\sum_{(s,t)\in S_{xy}} g_r(s,t)$	在S_{xy}中最低亮度的$d/2$和最高亮度和$d/2$被删除，$g_r(s,t)$表示在邻域中其余的$mn-d$个像素。用函数alphatrim实现（见本节中的常规函数spfilt）

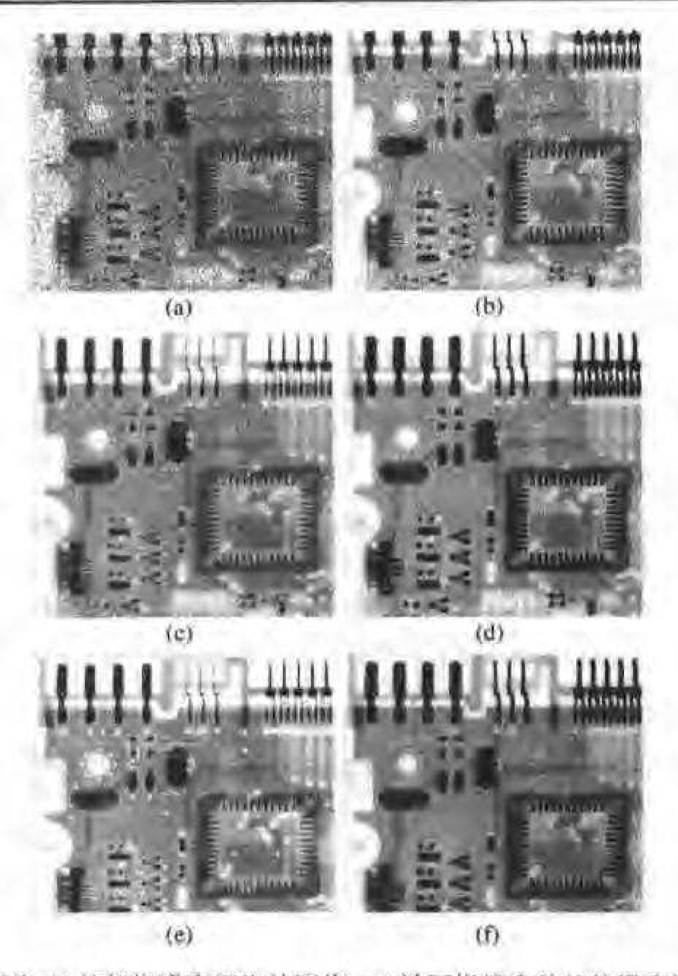


图5.5　(a)被概率为0.1的胡椒噪声污染的图像；(b)被同样概率的盐粒噪声污染的图像；(c)用阶为$Q = 1.5$的3 × 3反调和滤波器对(a)滤波的结果；(d)用$Q = -1.5$对(b)滤波的结果；(e)用3 × 3最大滤波器对(a)滤波的结果；(f)用3 × 3最小滤波器对(b)滤波的结果

5.3.2　自适应空间滤波器

前面一节讨论的滤波器适用于不考虑图像特性在不同位置之间的差异的图像。在有些应用中，可以通过使用能够根据被滤波区域的图像特性自适应的滤波器来改进结果。作为如何在MATLAB中实现自适应空间滤波器的说明，我们在本节中考虑一个自适应中值滤波器。如前面所述，S_{xy}表示一个将被处理的、中心在(x, y)处的子图像。在Gonzalez and Woods[2002]中详细说明的算法如下所示：

令

z_{min} 表示 S_{xy} 中的最小亮度值
z_{max} 表示 S_{xy} 中的最大亮度值
z_{med} 表示 S_{xy} 中的亮度中值
z_{xy} 表示坐标(x, y)处的亮度值

这个自适应中值滤波算法工作在两个层面，表示为level A和level B：

Level A：　若 $z_{min} < z_{med} < z_{max}$，则转向 level B
　　否则增加窗口尺寸
　　若窗口尺寸 $\leq S_{max}$，重复 level A
　　否则输出 z_{med}

Level *B*：　若 $z_{min} < z_{xy} < z_{max}$，则输出 z_{xy}
否则输出 z_{med}

其中，S_{max} 表示允许的最大自适应滤波器窗口的大小。Level *A* 最后一步的另一种选择是输出 z_{xy} 来代替中值。这将产生一个稍微清楚一些的结果，但是却可能探测不到与胡椒（盐粒）噪声值相同的、内含于常数背景中的盐粒（胡椒）噪声。

名为 adpmedian 的 M 函数可实现这个算法，它包含在附录 C 中。语法为

```
f = adpmedian(g, Smax)
```

其中，g 是将被滤波的图像，像上面定义的那样，Smax 是允许的最大自适应滤波器窗口的大小。

例 5.6　自适应中值滤波

图 5.6(a)显示了使用下面的命令产生的被椒盐噪声污染的电路板图像 f：

```
>> g = imnoise(f, 'salt & pepper', .25);
```

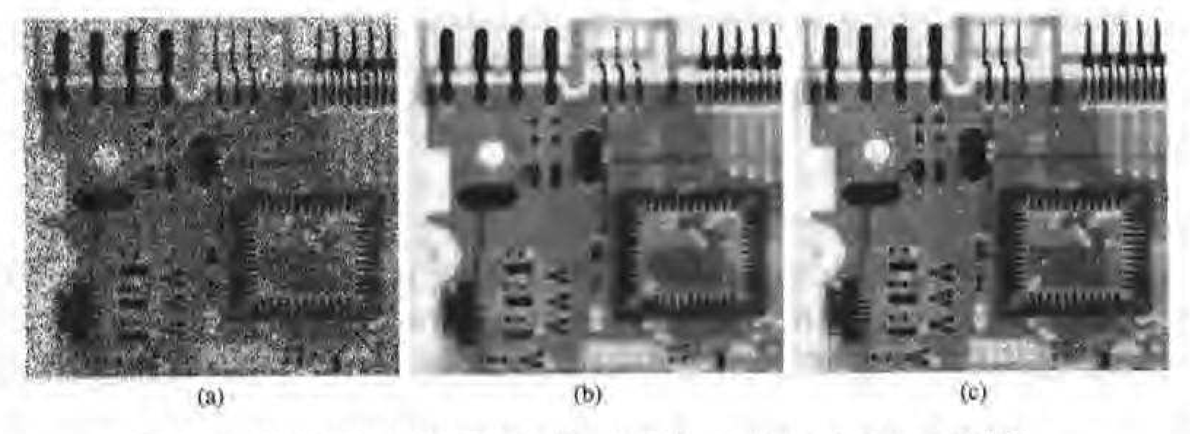
图 5.6(b)显示了使用下面的命令得到的结果（medfilt2 的使用请参看 3.5.2 节）：

```
>> f1 = medfilt2(g, [7 7], 'symmetric');
```

这幅图像确实没有噪声，但却非常模糊和失真（例如在图像中上部的接插键处）。另一方面，命令

```
>> f2 = adpmedian(g, 7);
```

产生图 5.6(c)所示的图像，它也确实没有噪声，但也明显地不像图 5.6(b)那样模糊和失真。



(a)　(b)　(c)

图 5.6　(a)被密度为 0.25 的椒盐噪声污染的图像；(b)使用大小为 7 × 7 的中值滤波器得到的结果；(c)使用 $S_{max} = 7$ 的自适应中值滤波得到的结果

5.4　通过频域滤波来降低周期噪声

如 5.2.3 节提到的那样，周期噪声本身表现为类似冲击的串，这种串在傅里叶频谱中通常是可见的。滤除这些成分的主要途径是通过陷波滤波。*n* 阶巴特沃兹陷波滤波器的传递函数由下式给出：

$$H(u,v) = \frac{1}{1 + \left[\dfrac{D_0^2}{D_1(u,v)D_2(u,v)}\right]^n}$$

其中，

$$D_1(u,v) = \left[(u - M/2 - u_0)^2 + (v - N/2 - v_0)^2\right]^{1/2}$$

和

$$D_2(u,v) = \left[(u - M/2 + u_0)^2 + (v - N/2 + v_0)^2\right]^{1/2}$$

其中，(u_0, v_0)和$(-u_0, -v_0)$是“陷波”的位置，D_0是它们的半径。注意，滤波器指定了频率矩形的中心，所以如4.2节和4.3节说明的那样，它必须在使用前用函数`fftshift`进行预处理。

为陷波滤波写一个M函数遵循与4.5节中用到的相同原理。写这个函数是较为可取的，以便可以输入多个陷波，正如5.2.3节中用来产生多个正弦噪声模式的方法一样。一旦得到了H，使用4.3.3节说明的函数`dftfilt`，滤波就完成了。

5.5 退化函数建模

当有类似于产生退化图像的设备时，通常通过做各种设备的设置实验来确定退化的本质是可能的。然而，相关的成像设备的可用性是解决图像复原问题的例外，而不是规则。典型的方法是通过产生PSF以及测试各种复原算法的结果来做实验。另一种方法是试图用数学方法把PSF模型化。这种方法不是这里讨论的主流；关于这个话题的介绍请参看Gonzalez and Woods[2002]。最后，若没有任何关于PSF的信息可用时，可以采取“盲去卷积”来推断PSF。这种方法将在5.10节中讨论。本节剩下的部分将集中讨论分别使用在3.4节和3.5节中介绍的函数`imfilter`和`fspecial`，以及在本章前面介绍的噪声生成函数来建模PSF的技术。

在图像复原问题中退到的一个主要的退化是图像模糊。由场景和传感器两者产生的模糊可以用空间域或频域的低通滤波器来建模。另一个重要的退化模型是在图像获取时传感器和场景之间的均匀线性运动而产生的图像模糊。我们可以使用IPT函数`fspecial`对图像模糊建模：

```
PSF = fspecial('motion', len, theta)
```

调用`fspecial`将返回PSF，它近似于由有着`len`个像素的摄像机的线性移动的效果。参数`theta`以度为单位，以顺时针方向对正水平轴度量。`len`的默认值是9，`theta`的默认值是0，在水平方向上它对应于9个像素的移动。

我们使用函数`imfilter`来创建一个已知`PSF`或用刚刚描述的方法计算得到的`PSF`的退化图像：

```
>> g = imfilter(f, PSF, 'circular');
```

其中，`'circular'`（见表3.2）用来减少边界效应。然后通过添加适当的噪声来构造退化的图像模型：

```
>> g = g + noise;
```

其中`noise`是一幅与`g`大小相同的随机噪声图像，它是使用5.2节中讨论的方法产生的。

当在给定的场合下比较本节和下面几节中讨论的各种方法的合理性时，使用相同的图像或测试图案是很有用的，这样比较才有意义。由函数`checkerboard`产生的测试图案对于实现这个目的非常有用，因为它的大小可以被缩放而不影响它的主要特征。语法为

```
C = checkerboard(NP, M, N)
```

其中，`NP`是每个正方形一边的像素数，`M`是行数，`N`是列数。若省略了`N`，则默认为`M`。若`M`和`N`都省略了，则将产生一个一面为8个正方形的正方形测试板。另外，若省略`NP`，则它将默认为10

个像素。测试板左半部分的亮正方形是白色的。测试板右半部分的亮正方形是灰色的。用下面的命令可产生一个亮正方形全是白色的测试板：

```
>> K = im2double(checkerboard(NP, M, N) > 0.5;
```

函数 checkerboard 产生的图像属 double 类图像，值在区间[0, 1]内。

由于有些复原算法对于大图像来说很慢，因此较好的方法是对小图像做实验来减少计算时间，并因此改善交互性。在这种情况下，若目的是显示，则通过像素复制来放大图像是很有用的。下面的函数用于实现该功能（代码请参见附录C）：

```
B = pixeldup(A, m, n)
```

该函数将A的每个像素在垂直方向上总共复制了m次，在水平方向上总共复制了n次。若省略了n，它将默认为m。

例 5.7　模糊噪声图像的建模

图 5.7(a)显示了一幅由下面的命令产生的测试板图像：

```
>> f = checkerboard(8);
```

图 5.7(b)所示的退化图像是使用下面的命令产生的：

```
>> PSF = fspecial('motion', 7, 45);
>> gb = imfilter(f, PSF, 'circular');
```

注意，PSF刚好是个空间滤波器。它的值为

```
>> PSF
PSF =
         0         0         0         0         0    0.0145         0
         0         0         0         0    0.0376    0.1283    0.0145
         0         0         0    0.0376    0.1283    0.0376         0
         0         0    0.0376    0.1283    0.0376         0         0
         0    0.0376    0.1283    0.0376         0         0         0
    0.0145    0.1283    0.0376         0         0         0         0
         0    0.0145         0         0         0         0         0
```

图 5.7(c)中的噪声模式是使用下面的命令产生的：

```
>> noise = imnoise(zeros(size(f)), 'gaussian', 0, 0.001);
```

通常，我们会直接使用imnoise(gb,'gaussian',0,0.001)将噪声加到gb上。然而，本章稍后将需要噪声图像，所以我们在此单独计算了它。

图 5.7(d)中的模糊噪声图像如下产生：

```
>> g = gb + noise;
```

在这幅图像中，噪声不容易看见，因为它的最大值为0.15，而图像的最大值是1。尽管如此，如5.7节和5.8节所示，这种噪声水平在试图复原g时却不是无关紧要的。最后，我们指出，图5.7中的所有图像都被放大到了512 × 512，并且是使用下面的命令形式来显示的：

```
>> imshow(pixeldup(f, 8), [ ])
```

图 5.7(d)中的图像将在例5.8和例5.9中复原。

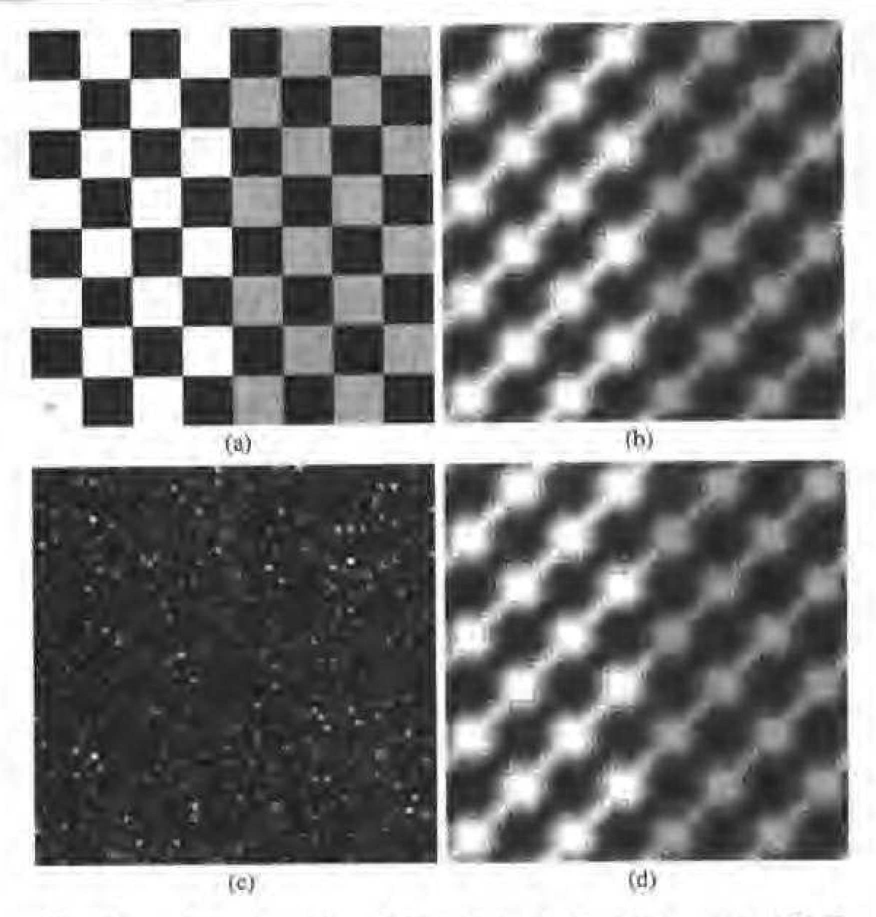


图 5.7 (a)原图像；(b)使用参数为 `len = 7` 和 `theta = -45` 度的函数 `fspecial` 模糊的图像；(c)噪声图像；(d)图(b)和图(c)之和

5.6 直接逆滤波

用于复原一幅退化图像的最简方法是构成一个如下形式的估计：

$$\hat{F}(u,v) = \frac{G(u,v)}{H(u,v)}$$

然后采用$\hat{F}(u,v)$[回忆$G(u,v)$退化图像的傅里叶变换]的傅里叶逆变换来得到图像的相应估计。这种方法称为逆滤波。由5.1节讨论的模型，我们可以将估计表示为

$$\hat{F}(u,v) = F(u,v) + \frac{N(u,v)}{H(u,v)}$$

这个简单的表达式告诉我们，即使准确地知道了$H(u,v)$，也不能恢复$F(u,v)$[因此也不能恢复原始的、未被退化的图像$f(x,y)$]，因为噪声分量是一个随机函数，它的傅里叶变换$N(u,v)$是未知的。另外，在实际中，有许多$H(u,v)$为零的情况也是个问题。即使$N(u,v)$项可以忽略，用为零的$H(u,v)$值来除它也将会控制复原估计。

采用逆滤波时，典型的方法是形成比率$\hat{F}(u,v) = G(u,v)/H(u,v)$，然后，为了得到逆，将频率的范围限制在接近原点的频率。概念是$H(u,v)$中的零不太可能在接近原点的地方出现，因为变换数值通常是该区域中的最高值。这个基本主题有很多变化，其中，使H为零或接近零的(u,v)值被特殊对待。这种方法有时称为伪逆滤波。通常，如例5.8所示，基于这种类型的逆滤波方法很少有使用价值。

5.7 维纳滤波

维纳滤波（N. Wiener最先在1942年提出的方法）是一种最早也最为人们熟知的线性图像复原方法。维纳滤波器寻找一个使统计误差函数

$$e^2 = E\{(f - \hat{f})^2\}$$

最小的估计$\hat{f}$。其中，E是期望值操作符，f是未退化的图像。该表达式在频域可表示为

$$\hat{F}(u,v) = \left[\frac{1}{H(u,v)}\frac{|H(u,v)|^2}{|H(u,v)|^2 + S_\eta(u,v)/S_f(u,v)}\right]G(u,v)$$

其中，

$H(u,v)$表示退化函数
$|H(u,v)|^2 = H^*(u,v)H(u,v)$
$H^*(u,v)$表示$H(u,v)$的复共轭
$S_\eta(u,v) = |N(u,v)|^2$表示噪声的功率谱
$S_f(u,v) = |F(u,v)|^2$表示未退化图像的功率谱

比率$S_\eta(u,v) / S_f(u,v)$称为噪信功率比。我们看到，若对于u和v的所有相关值，噪声功率谱为零，则这个比率就变为零，且维纳滤波器就成为前一节中讨论的逆滤波器。

我们感兴趣的两个量为平均噪声功率和平均图像功率，分别定义为

$$\eta_A = \frac{1}{MN}\sum_u \sum_v S_\eta(u,v)$$

和

$$f_A = \frac{1}{MN}\sum_u \sum_v S_f(u,v)$$

其中，M和N分别表示图像和噪声数组的垂直和水平大小。这些量都是标量常量，且它们的比率

$$R = \frac{\eta_A}{f_A}$$

也是一个标量，有时用来代替函数$S_\eta(u,v) / S_f(u,v)$，以便产生一个常量数组。在这种情况下，即使真实的比率未知，交互式地变化常量并观察复原的结果的实验就变成了一件简单的事。当然，假设函数为常量是一种粗糙的近似。在前述的滤波器方程中，用一个常量数组来代替$S_\eta(u,v) / S_f(u,v)$就产生了所谓的参数维纳滤波器。如例5.8所示，使用一个常量数组的简单行为可以对直接逆滤波产生重大的改进。

在IPT中，维纳滤波是使用函数`deconvwnr`来实现的，函数`deconvwnr`有三种可能的语法形式。在所有的这些形式中，`g`代表退化图像，`fr`是复原图像。第一种语法形式

```
fr = deconvwnr(g, PSF)
```

假设噪信比为零。从而，维纳滤波器的这种形式就是5.6节中提到的逆滤波器。语法

```
fr = deconvwnr(g, PSF, NSPR)
```

假设噪信功率比已知，或是个常量或是个数组；函数接受其中的任何一个。这是用于实现参数维纳滤波器的语法，在这种情况下，NSPR可以是一个交互的标量输入。最后，语法

个问题变得更复杂了。尽管如此，将复原问题用矩阵的形式加以公式化表达确实简化了复原技术的推导。

虽然到现在为止我们还没有推导约束的最小二乘方方法，但这种方法的核心是在前面的段落中提到的有关**H**的逆的敏感问题。处理这个问题的一种方法是基于平滑度测量的复原最优性，例如，图像的二阶导数(即拉普拉斯算子)。为有实际意义，复原过程必须用我们手中的参数加以约束。因此，需要寻找准则函数C的最小值，函数C定义为

$$C = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\nabla^2 f(x,y)]^2$$

该函数的约束条件为

$$\|\mathbf{g} - \mathbf{H}\hat{\mathbf{f}}\|^2 = \|\boldsymbol{\eta}\|^2$$

其中，$\|\mathbf{w}\|^2 \triangleq \mathbf{w}^T\mathbf{w}$是欧几里得向量范数①，$\hat{\mathbf{f}}$是非退化图像的估计，拉普拉斯算子$\nabla^2$的定义见3.5.1节。

这个最优化问题的频域解决办法由下式给出：

$$\hat{F}(u,v) = \left[\frac{H^*(u,v)}{|H(u,v)|^2 + \gamma|P(u,v)|^2}\right]G(u,v)$$

其中，γ是一个必须加以调整的参量，以便约束条件得到满足(若γ为0，则会得到逆滤波方案)，$P(u,v)$是函数

$$p(x,y) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

的傅里叶变换。我们将这个函数称为拉普拉斯算子，见3.5.1节。在前述方程中，未知量只有γ和$\|\boldsymbol{\eta}\|^2$两个。然而，我们又可以看出，若已知与噪声功率(标量)成比例的$\|\boldsymbol{\eta}\|^2$，则通过迭代，γ便可以得出。

约束的最小二乘方滤波在IPT中是通过函数`deconvreg`来实现的，其语法为

```
fr = deconvreg(g, PSF, NOISEPOWER, RANGE)
```

其中，`g`是被污染的图像，`fr`是复原的图像，`NOISEPOWER`与$\|\boldsymbol{\eta}\|^2$成比例，`RANGE`为值的范围，在求解γ时，该算法受这个范围的限制。默认范围是$[10^{-9}, 10^9]$(MATLAB中的符号为`[1e-10,1e10]`)。若将上述两个参数排除在参量之外，则函数`deconvreg`就会产生一个逆滤波方案。`NOISEPOWER`的较好初始估计为$MN[\sigma_\eta^2 + m_\eta^2]$，其中$M$和$N$代表图像的维数，括号中的参量代表噪声方差和噪声均值的平方。该估计只是一个简单的开始，如下面的例子中所显示的那样，最后的结果值可能会有很大的不同。

例5.9 使用函数`deconvreg`复原模糊噪声图像

现在利用函数`deconvreg`复原图5.7(d)所示的图像，该图像的大小为64×64，且由例5.7可知，噪声的方差为0.001，均值为0。所以，`NOISEPOWER`的初始估计为$(64)^2[0.001-0] \approx 4$。图5.9(a)显示了利用如下命令得出的结果：

```
>> fr = deconvreg(g, PSF, 4);
```

① 对于具有n个分量的列向量$\mathbf{w}$，$\mathbf{w}^T\mathbf{w} = \sum_{k=1}^{n} w_k^2$，其中$w_k$是$\mathbf{w}$的第$k$个分量。

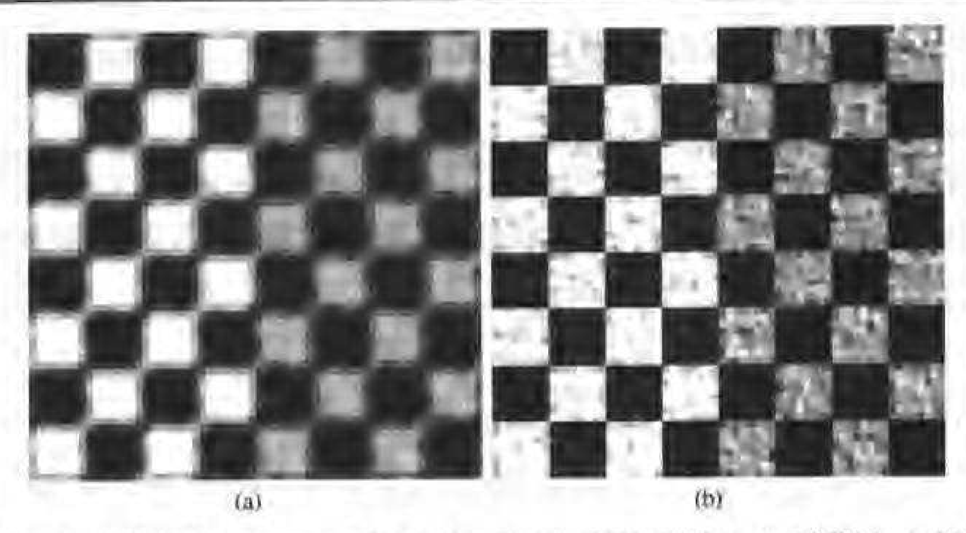


图 5.9 (a)使用参量NOISEPOWER等于4的正则滤波器复原的图5.7(d)中的图像；(b)使用参量 NOISEPOWER 为0.4 和RANGE 为[1e-7 1e7]的正则滤波器复原的同一幅图像

其中，g和PSF都来自于例5.7。该图像与原图像相比已有改善，但也可以明显地看出NOISEPOWER的值并不是非常好。在对这个参数和参数RANGE进行实验之后，得出如图5.9(b)所示的结果，它是使用如下命令得到的：

```
>>fr = deconvreg(g, PSF, 0.4, [1e-7  1e7]);
```

这样，我们就需要把NOISEPOWER的值的量级下调一个等级，而RANGE比默认值更加紧缩。图5.8(d)所示的Wiener滤波结果要好得多，但条件是我们必须已经对噪声和图像谱的知识有足够的了解。若没有这些信息，则用两种滤波器通过实验得到的结果常常是可比的。

若复原图像呈现由算法中使用的离散傅里叶变换所引入的振铃，则在调用函数deconvreg之前使用函数 edgetaper 是有帮助的（见5.7节）。

5.9 使用 Lucy-Richardson 算法的迭代非线性复原

前三节中所讨论过的图像复原方法都是线性的。在感觉上它们也更“直接”，因为复原滤波一旦被指定下来，相应的解决办法就会通过滤波器的应用得到。这种实现的简单性、适量的运算要求和容易建立的理论基础，使得线性方法在很多年间都是图像复原的一个基本工具。

在过去的20年里，非线性迭代技术已经越来越多地被人们接受，作为复原的工具，它常常会获得比线性方法更好的结果。非线性方法的主要缺陷是它们的特性常常是不可预见的，并且它们常常需要重要的计算资源。现在第一个缺陷已经不太重要了，因为在很多应用领域非线性技术都优于线性技术（Jansson[1997]）。第二个缺陷也不再是个问题，因为在过去的十年中计算能力一直在以惊人的速度增长。在工具箱中选择的非线性方法是由Richardson[1972]和Lucy[1974]独立开发的技术。工具箱提供的这些算法被称为Lucy-Richardson（L-R）算法，在一些文献中它又被称为Richardson-Lucy算法。

L-R算法是从最大似然公式中引出来的（见5.10节），在这个方程中，图像是用泊松统计加以模型化的。当下面这个迭代收敛时，模型的最大似然函数可以得到一个令人满意的方程：

$$\hat{f}_{k+1}(x,y) = \hat{f}_k(x,y)\left[h(-x,-y) * \frac{g(x,y)}{h(x,y) * \hat{f}_k(x,y)}\right]$$

如前边一样，“*”代表卷积，$\hat{f}$是未退化图像的估计，g和h都与5.1节中所定义的相同。这个算法的迭代本质是显而易见的。它的非线性本质是在方程右边用$\hat{f}$来除产生的。

就像大多数非线性方法一样，关于什么时候停止L-R算法通常很难回答。通常，接下来的处理途径是对于给定的应用，在我们获得满意的结果时，观察输出并终止算法。

在IPT中，L-R算法是由名为deconvlucy的函数完成的，该函数语法为

```
fr = deconvlucy(g, PSF, NUMIT, DAMPAR, WEIGHT)
```

其中，fr代表复原的图像，g代表退化的图像，PSF是点扩散函数，NUMIT为迭代的次数（默认为10次），DAMPAR和WEIGHT定义如下。

DAMPAR是一个标量，它指定了结果图像与原图像g之间的偏离阈值。当像素值偏离原值的范围在DAMPAR之内时，就不用再迭代。这既抑制了这些像素上的噪声，又保存了必要的图像细节。默认值为0（无衰减）。

WEIGHT是一个与g同样大小的数组，它为每一个像素分配一个权重来反映其重量。例如从某个有缺陷的成像数组得出的一个不良像素最终会被赋以权重值零，从而排除该像素来求解。这个数组的另一个用处是可以根据平坦区域修正量调整像素的权重，根据成像数组的知识，这一点是必要的。当用一个指定的PSF来模拟模糊时（见例5.7），WEIGHT可以从计算像素中剔除那些来自图像边界的像素点，因此，PSF造成的模糊是不同的。若PSF的大小为$n \times n$，则在WEIGHT中用到的零边界的宽度是ceil(n/2)。默认值是同输入图像g同等大小的一个单位数组。

若复原图像呈现出由算法中所用的离散傅里叶变换所引入的振铃，则在调用函数deconvlucy之前，要先利用函数edgetaper（见5.7节）。

例5.10 使用函数deconvlucy复原模糊噪声图像

图5.10(a)示出了一幅用如下命令产生的图像：

```
>> f = checkerboard(8);
```

该命令产生一幅大小为64 × 64像素的方形图像。与前面一样，出于显示的目的，我们用函数pixeldup来将它的大小增大为512 × 512：

```
>> imshow(pixeldup(f, 8));
```

如下命令产生一个大小为7 × 7且标准偏差为10的高斯PSF：

```
>> PSF = fspecial('gaussian', 7, 10);
```

接下来，用PDF模糊图像f，在其上添加均值为0、标准偏差为0.01的高斯噪声：

```
>> SD = 0.01;
>> g = imnoise(imfilter(f, PSF), 'gaussian', 0, SD^2);
```

图5.10(b)显示了结果。

这个例子的余下部分使用函数deconvlucy来对图像g进行复原处理。我们将DAMPAR赋以10倍于SD的值：

```
>> DAMPAR = 10*SD;
```

数组WEIGHT用我们前面讨论的办法产生：

```
>> LIM = ceil(size(PSF, 1)/2);
>> WEIGHT = zeros(size(g));
>> WEIGHT(LIM + 1:end - LIM, LIM + 1:end - LIM) = 1;
```

WEIGHT数组的大小是 64×64，并且有值为0的4像素宽的边界，其余的像素都是1。

惟一剩下的变量是NUMIT，即迭代的次数。图5.10(c)示出了使用这个命令之后获得的结果。

```
>> NUMIT = 5;
>> f5 = deconvlucy(g, PSF, NUMIT, DAMPAR, WEIGHT);
>> imshow(pixeldup(f5, 8))
```

虽然图像已经稍微有些改进了，但是仍旧模糊。图5.10(d)和图5.10(e)显示了应用NUMIT = 10和20后得到的结果。其中后一幅图像是对模糊和噪声图像的合理复原。事实上，进一步增加迭代次数在复原结果上并没有显著的改进。例如，图5.10(f)是使用了100次迭代后获得的结果。这幅图像只是比使用了20次迭代后获得的图像稍稍清晰和明亮了一些。在所有结果中，细黑色边界都是由数组WEIGHT中的0所引起的。

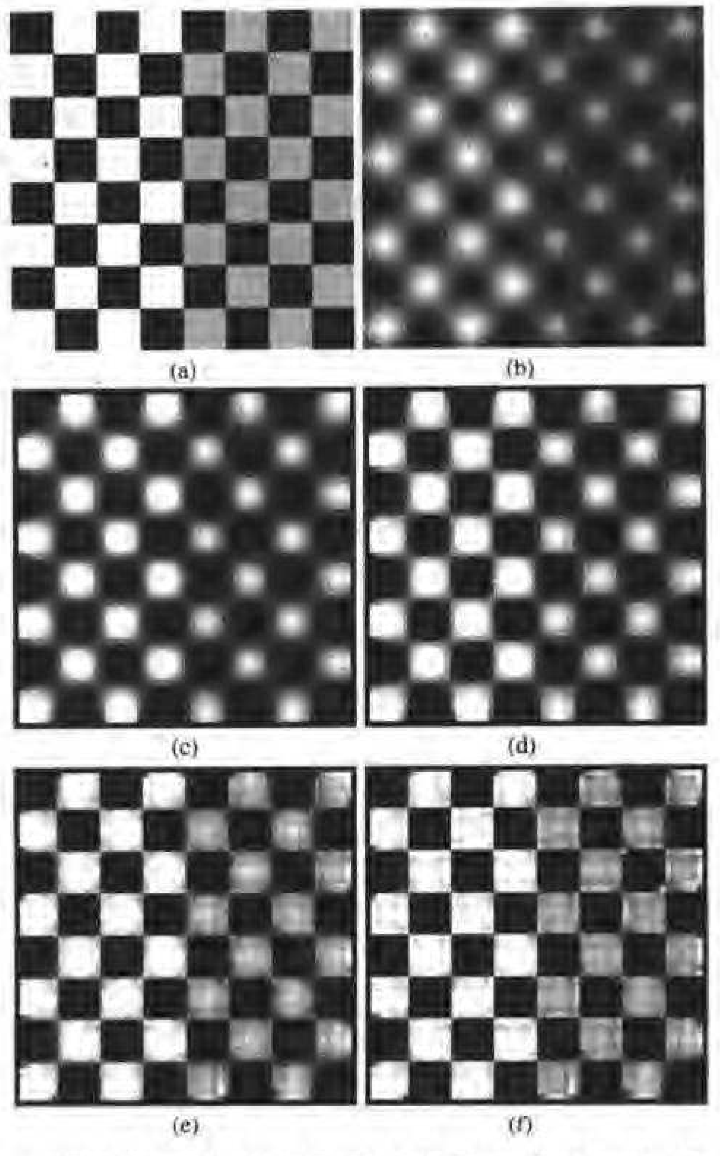


图5.10　(a)原图像；(b)由高斯噪声污染和模糊的图像；(c)到(f)对图像(b)用L-R算法分别迭代5次、10次、20次和100次后的复原图像

5.10 盲去卷积

在图像复原过程中，最困难的问题之一是，如何像前面几节讨论的那样获得复原算法中使用的PSF的恰当估计。正如先前表明过的那样，那些不以PSF知识为基础的图像复原方法统称为盲去卷积算法。

在过去的20年里，一种盲去卷积的方法已经受到了人们的极大重视，它是以最大似然估计（MLE）为基础的，即一种用被随机噪声所干扰的量进行估计的最优化策略。简要地说，关于MLE方法的一种解释就是将图像数据看成是随机量，它们与另外一族可能的随机量之间有着某种似然性。似然函数用 $g(x, y)$、$f(x, y)$和 $h(x, y)$来加以表达（见5.1节），然后，问题就变成了寻求最大似然函数。在盲去卷积中，最优化问题用规定的约束条件并假定收敛时通过迭代来求解，得到的最大$f(x, y)$和 $h(x, y)$就是还原的图像和PSF。

关于盲去卷积MLE的推导已超出了我们所讨论的范围，但是读者可以通过参考以下书目进一步加深自己的理解；最大似然估计的背景，可参考Van Trees[1968]的经典书籍；对图像处理领域中的原始工作的综述，可参考Dempster等人[1977]的论著；关于进一步的进展，可参考Holmes[1992]的著作。去卷积的综合性参考书由Jansson[1997]所著；一些去卷积方法在显微技术上和天文学上应用的详细例子，可以分别参考Holmes等人[1995]和Hanisch等人[1997]的著作。

工具箱通过函数 deconvblind 来执行盲去卷积，它有如下语法：

```
[f, PSFe] = deconvblind(g, INITPSF)
```

其中，g代表退化图像，INITPSF是点扩散函数的初始估计。PSFe是这个函数最终计算得到的估计值，fr是利用估计的PSF复原的图像。用来取得复原图像的算法是5.9节中描述过的L-R迭代复原算法。PSF估计受其初始推测尺寸的巨大影响，而很少受其值的影响（一个值为1的数组是合理的初始推测）。

在前面这个语法中，迭代的次数默认为10次。这个函数中包含的附加参数用来控制迭代的次数和复原的其他特性，正如下面的语法所示：

```
[fr, PSFe] = deconvblind(g, INITPSF, NUMIT, DAMPAR, WEIGHT)
```

其中NUMIT, DAMPAR和WEIGHT的描述见前节中的L-R算法。

若复原图像呈现出由算法中使用的离散傅里叶变换所引入的振铃，则我们在调用函数deconvblind之前，通常要使用函数edgetaper（见5.7节）。

例5.11 使用函数deconvblind估计PSF

图5.11(a)就是生成图5.10(b)所示退化图像的PSF：

```
>> PSF = fspecial('gaussian', 7, 10);
>> imshow(pixeldup(PSF, 73), [ ]);
```

如在例5.10中，问题中的退化图像是用如下命令得到的：

```
>> SD = 0.01;
>> g = imnoise(imfilter(f, PSF), 'gaussian', 0, SD^2);
```

在当前这个例子中，我们感兴趣的是仅从给出的退化图像g便可以利用函数deconvblind获得PSF的估计值。图5.11(b)显示了由如下命令得出的PSF：

```
>> INITPSF = ones(size(PSF));
>> NUMIT = 5;
```

```
>> [fr, PSFe] = deconvblind(g, INITPSF, NUMIT, DAMPAR, WEIGHT);
>> imshow(pixeldup(PSFe, 73), [ ]);
```

其中 DAMPAR 和 WEIGHT 的取值与例 5.10 中的取值相同。

图 5.11(c)和图 5.11(d)都是用与 PSFe 相同的方式加以显示的，它们示出了分别利用 10 次迭代和 20 次迭代后所得到的 PSF，后者的结果更接近于图 5.11(a)中的真正 PSF。

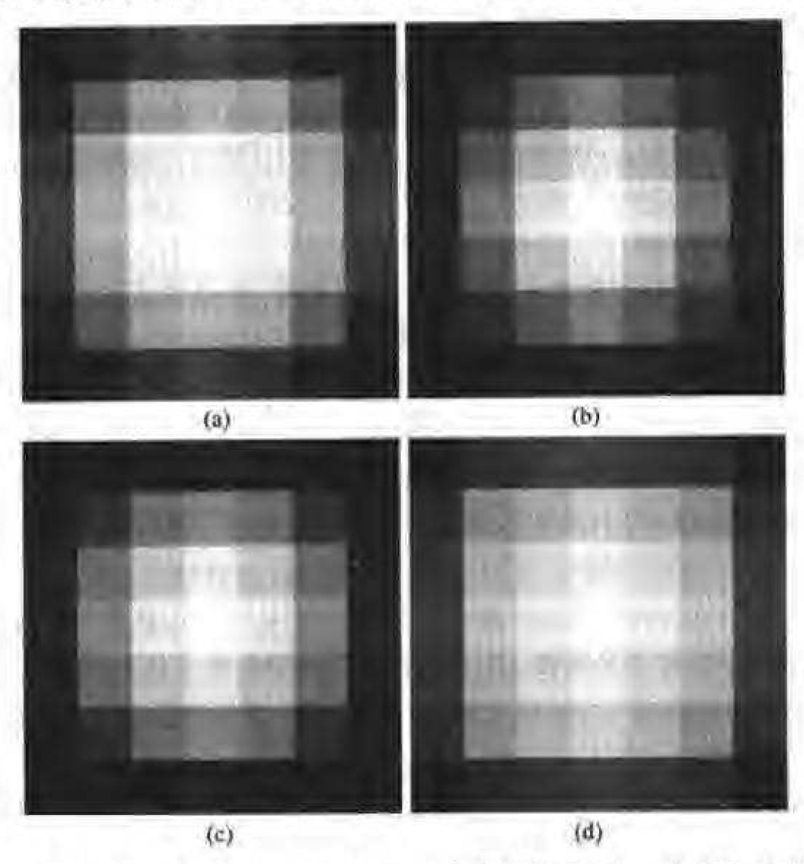


图 5.11 (a)原始 PSF；(b)到(d)在函数 deconvblind 中分别使用 5 次、10 次和 20 次迭代估计的 PSF

5.11 几何变换与图像配准

我们以介绍图像复原中涉及到的几何变换来结束本章的内容。几何变换会对图像的像素之间的空间关系加以修改。它们常常称为"橡皮布变换"，因为看起来这种方法就像在一块橡皮布上印下一张图，然后根据预先确定的一套规则将这张布加以拉伸。

几何变换常常用来实现图像配准。图像配准就是取两张相同场景的图像并加以对准，从而可以将它们合并，以便目测或者定量比较。在随后的这一节中，我们将讨论：(1)空间变换以及如何定义它们并在MATLAB中直观显示，(2)怎样把空间变换应用于图像，(3)在图像配准中如何确定空间变换。

5.11.1 空间几何变换

假设有一幅定义在(w, z)坐标系上的图像f经过几何变形后产生了定义在(x, y)坐标系上的图像g。这个（坐标系的）变换可以表示为

$$(x, y) = T\{(w, z)\}$$

例如，若$(x, y) = T\{(w, v)\} = (w/2, z/2)$，则这个"变形"其实仅仅就是图像$f$的大小在空间维数上都缩小一半，如图 5.12 所示。

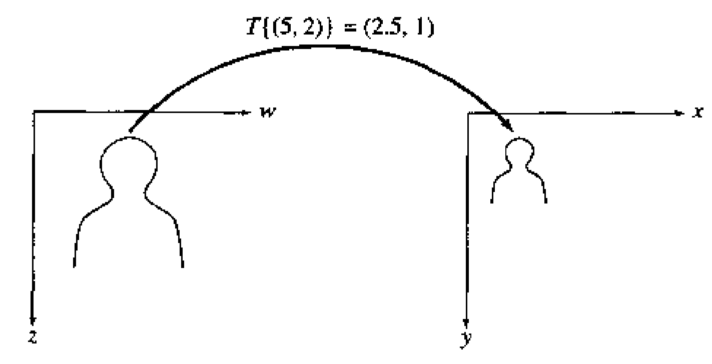


图5.12 一个简单的空间变换（注意，此图中的 xy 轴与2.1.1节中定义的图像轴并不对应。如在那一节中提及的那样，IPT有时会用到所谓的空间坐标系，并用 y 表示行、x 表示列。为使几何变换的表示与IPT文件的表示一致，在本节中我们沿用这种坐标系）

仿射变换是最常用的一种空间变换形式（Wolberg[1990]）。仿射变换可以用如下矩阵形式表示：

$$[x \quad y \quad 1] = [w \quad z \quad 1]\mathbf{T} = [w \quad z \quad 1]\begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}$$

该变换可以按比例缩放、旋转、平移或者剪切一些点等，具体取决于 **T** 的元素的取值。表5.3示出了怎样选择不同的 **T** 值以完成不同的变换。

表5.3 仿射变换的类型

类型	仿射矩阵 T	坐标方程	图示
单位	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = w$ $y = z$	
缩放	$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = s_x w$ $y = s_y z$	
旋转	$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = w\cos\theta - z\sin\theta$ $y = w\sin\theta + z\cos\theta$	
水平剪切	$\begin{bmatrix} 1 & 0 & 0 \\ \alpha & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = w + \alpha z$ $y = z$	
垂直剪切	$\begin{bmatrix} 1 & \beta & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = w$ $y = \beta w + z$	
平移	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \delta_x & \delta_y & 1 \end{bmatrix}$	$x = w + \delta_x$ $y = z + \delta_y$	

IPT 利用一个所谓的 tform 结构来表示空间变换。创建这种结构的一种方法是利用函数 maketform，它的调用语法如下所示：

```
tform = maketform(transform_type, transform_parameters)
```

第一个输入变量 transform_type 是下列字符串之一：'affine'、'projective'、'box'、'composite' 或 'custom'。5.11.3 节中的表 5.4 对这些变换类型给予了论述。其他依赖于变换类型的附加参量会在函数 maketform 的帮助文件中加以详细说明。

在本节中，我们的兴趣在仿射变换上。例如，创建仿射变换 tform 的一种途径就是直接提供矩阵 **T**

```
>> T = [2 0 0; 0 3 0; 0 0 1];
>> tform = maketform('affine', T)
tform =
       ndims_in: 2
      ndims_out: 2
    forward_fcn: @fwd_affine
    inverse_fcn: @inv_affine
          tdata: [1 x 1 struct]
```

虽然没有必要直接应用 tform 结构的域，但 **T** 和 $\mathbf{T}^{-1}$ 的信息已包含在 tdata 域中：

```
>> tform.tdata

ans =
       T: [3 x 3 double]
    Tinv: [3 x 3 double]

>> tform.tdata.T
ans =
    2    0    0
    0    3    0
    0    0    1
>> tform.tdata.Tinv
ans =
    0.5000         0         0
         0    0.3333         0
         0         0    1.0000
```

IPT 提供两个用于对点进行空间变换的函数：tformfwd 计算正变换 $T\{(w, z)\}$，tforminv 计算逆变换 $T^{-1}\{(x, y)\}$。tformfwd 的调用语法为 XY = tformfwd(WZ,tform)，其中 WZ 是一个大小为 $P \times 2$ 的点矩阵。WZ 的每一行都包含一个点的 w 坐标和 z 坐标。类似地，XY 是一个大小为 $P \times 2$ 的点矩阵，每一行都包含变换点的 x 和 y 坐标。例如，下面的命令先计算出一对点的正变换，紧接着做一个逆变换，以验证返回的原始数据：

```
>> WZ = [1 1; 3 2];
>> XY = tformfwd(WZ, tform)
XY =
    2    3
    6    6
>> WZ2 = tforminv(XY, tform)
WZ2 =
    1    1
    3    2
```

为了更好地感受特定空间变换的效果，观察它是怎样对一组栅格排列的点做变换是很有用的。下面的M函数vistformfwd建立点的栅格结构，使用函数tformfwd对栅格进行变换，然后将原来的栅格和变换后的栅格紧挨着画在一起，以便比对。注意，为创建栅格，函数meshgrid（见2.10.4节）和linspace（见2.8.1节）已联合使用。下面的代码也说明了在本节中至今为止讨论过的一些函数的使用。

```
function vistformfwd(tform, wdata, zdata, N)
%VISTFORMFWD Visualize forward geometric transform.
%   VISTFORMFWD(TFORM, WRANGE, ZRANGE, N) shows two plots: an N-by-N
%   grid in the W-Z coordinate system, and the spatially transformed
%   grid in the X-Y coordinate system.  WRANGE and ZRANGE are
%   two-element vectors specifying the desired range for the grid. N
%   can be omitted, in which case the default value is 10.

if nargin < 4
   N = 10;
end
% Create the w-z grid and transform it.
[w, z] = meshgrid(linspace(wdata(1), zdata(2), N), ...
                  linspace(wdata(1), zdata(2), N));
wz = [w(:) z(:)];
xy = tformfwd([w(:) z(:)], tform);

% Calculate the minimum and maximum values of w and x,
% as well as z and y. These are used so the two plots can be
% displayed using the same scale.
x = reshape(xy(:, 1), size(w)); % reshape is discussed in Sec. 8.2.2.
y = reshape(xy(:, 2), size(z));
wx = [w(:); x(:)];
wxlimits = [min(wx) max(wx)];
zy = [z(:); y(:)];
zylimits = [min(zy) max(zy)];

% Create the w-z plot.
subplot(1,2,1) % See Section 7.2.1 for a discussion of this function.
plot(w, z, 'b'), axis equal, axis ij
hold on
plot(w', z', 'b')
hold off
xlim(wxlimits)
ylim(zylimits)
set(gca, 'XAxisLocation', 'top')
xlabel('w'), ylabel('z')

% Create the x-y plot.
subplot(1, 2, 2)
plot(x, y, 'b'), axis equal, axis ij
hold on
plot(x', y', 'b')
hold off
xlim(wxlimits)
ylim(zylimits)
set(gca, 'XAxisLocation', 'top')
xlabel('x'), ylabel('y')
```

$$\mathbf{T}=\begin{bmatrix} s\cos\theta & s\sin\theta & 0 \\ -s\sin\theta & s\cos\theta & 0 \\ \delta_x & \delta_y & 1 \end{bmatrix}$$

下面的命令产生一个等角变换并用于测试图像：

```
>> f = checkerboard(50);
>> s = 0.8;
>> theta = pi/6;
>> T = [s*cos(theta) s*sin(theta) 0
       -s*sin(theta) s*cos(theta) 0
        0    0    1];
>> tform = maketform('affine', T);
>> g = imtransform(f, tform);
```

图5.14(a)和图5.14(b)显示了原始的和变换后的测试板图像。

前面对函数imtransform的调用使用了默认的插值方式'bilinear'。如前面介绍的那样，我们也可以选择一种不同的内插方法，如最近邻法，方法是明确地指定imtransform的调用方式：

```
>> g2 = imtransform(f, tform, 'nearest');
```

图5.14(c)显示了结果。最近邻内插比双线性内插方式选度更快，而且在很多场合它都更合理，但它产生的结果却往往比双线性方法差。

函数imtransform有一些额外的可选参数，它们往往也是很有用的。例如，赋给它一个参数FillValue，可以控制imtransform用于输入图像区域之外的像素颜色：

```
>> g3 = imtransform(f, tform, 'FillValue', 0.5);
```

在图5.14(d)中，原图像外部的像素已用中等的灰度取代了黑色。

其他的附加变量可以帮助解决在利用imtransform平移图像时常常引发的一些混乱。例如，下面的命令完成了一个纯平移：

```
>> T2 = [1 0 0; 0 1 0; 50 50 1];
>> tform2 = maketform('affine', T2);
>> g4 = imtransform(f, tform2);
```

这个命令得到的结果与图5.14(a)中的原图像是一致的。

这种效果是由imtransform的默认特性决定的。imtransform决定输出图像在输出坐标系中的边框（见11.4.1节中有关边框的介绍）。默认情况下，在边框上只进行反向映射变换。这种方法可以有效地撤销平移。通过详细指定参量XData和YData的值，可以确切地指定imtransform在哪个输出空间上计算输出结果。XData是一个包含两个元素的向量，指定了输出图像的最左列和最右列的位置；YData也是一个包含两个元素的向量，它指定了输出图像最顶部行和最底部行的位置。下面的这个命令在$(x, y) = (1, 1)$和$(x, y) = (400, 400)$之间的区域内计算输出图像：

```
>> g5 = imtransform(f, tform2,'XData', [1 400], 'YData', [1 400], ...
                    'FillValue', 0.5);
```

图5.14(e)显示出了计算结果。

函数 imtransform 的其他设定以及相关的 IPT 函数为计算结果提供了附加控制操作，尤其体现在插值过程的实现上。很多有关工具箱的说明文献都放在函数 imtransform 和 makeresampler 的帮助页中。

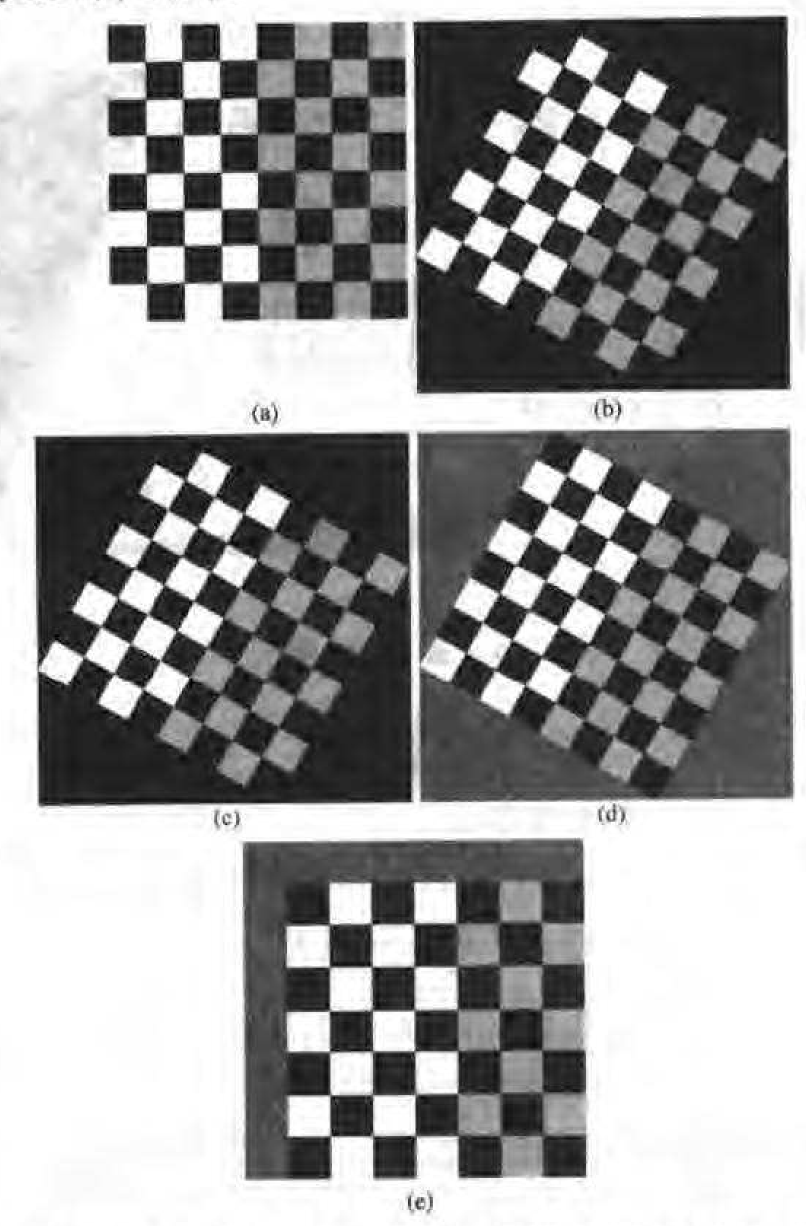


图 5.14 测试板图像的仿射变换：(a)原图像；(b)使用默认内插（双线性）方法的线性等角变换后的图像；(c)使用最近邻内插方法后的图像；(d)指定另一个填充值后的图像；(e)控制输出空间位置，以便平移是可见的

5.11.3 图像配准

图像配准方法寻求将两幅相同场景的图像加以对准。例如，将两幅或者更多幅大致在同一时间拍摄但由不同设备完成的图像对准，如一幅 MRI（磁共振成像）扫描图像，一幅 PET（正电子发射成像）扫描图像。或者，图像可能是在不同时间用同一台设备采集的，如卫星图片或在同一指定地点相隔几天、几个月甚至几年时间采集的图像。无论是上述哪种情况，合并这些图像或者做定量的分析比较都需要对可能出现的几何误差进行补偿。导致这些误差出现的原因有很多，如摄像机的角度、距离和方向，物体位置的移动以及其他因素。

工具箱支持以控制点为基础的图像配准，控制点也称为联结点，即在两幅图像中的位置已知或可以交互地选择的像素的一个子集。图5.15利用一种测试图案和这种图案的一个已经过投影变形的版本说明了控制点的概念。一旦选出了足够的有效控制点，就可以利用IPT函数 cp2tform 去匹配一个指定了类型的空间变换，以便控制这些点。函数cp2tform支持的空间变换类型已列在表5.4中。

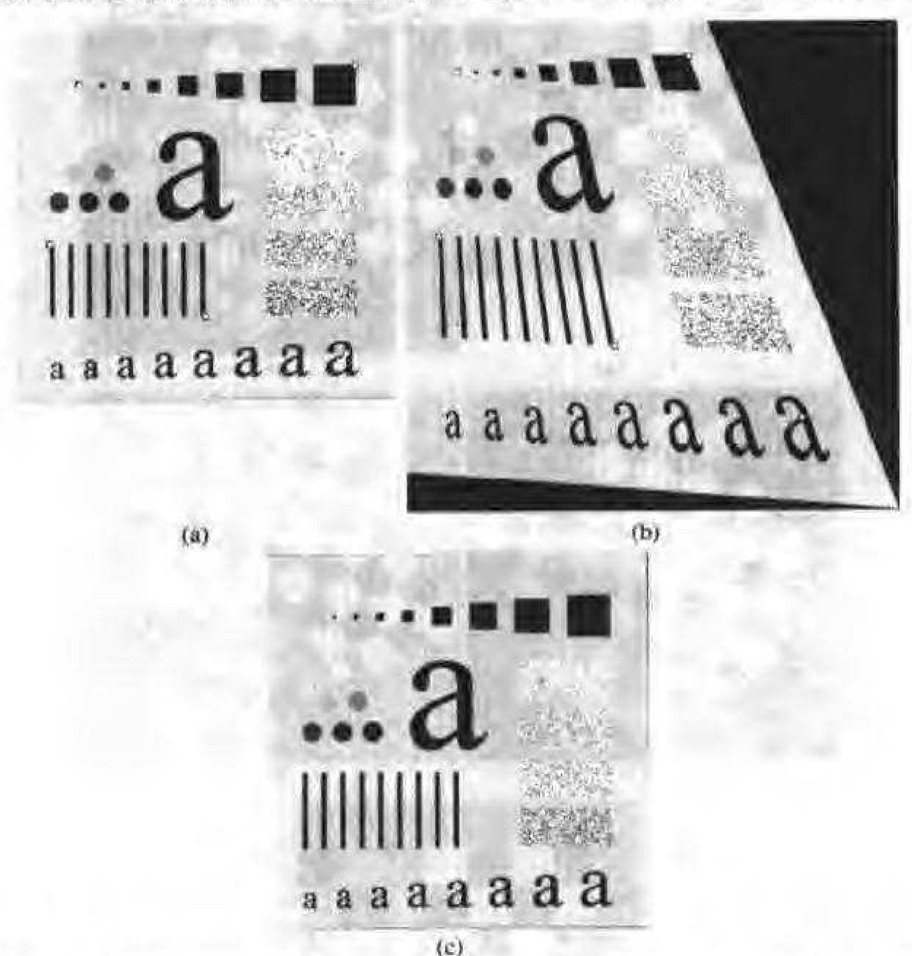


图5.15 基于控制点的图像配准：(a)具有控制点（叠加在图像上的小圆圈）的图像；(b)具有控制点的失真图像；(c)使用由控制点推导的投影变换后的校正图像

表5.4 函数 cp2tform 和 maketform 支持的变换类型

变换类型	描述	函数
Affine（仿射）	缩放、旋转、剪切和平移的组合。直线仍保持为直线，平行线保持为平行线	maketform cp2tform
Box（盒）	沿每个方向独立地缩放和平移；仿射变换的子集	maketform
Composite（合成）	顺序应用的空间变换的集合	maketform
Custom（自定义）	用户定义的空间变换；用户提供T和T^{-1}函数	maketform
Linear conformal（线性等角）	缩放（在各个方向上都相同）、旋转和平移；仿射变换的子集	cp2tform
LWM	局部加权平均；局部变化的空间变换	cp2tform
Piecewise linear（分段线性）	局部变化的空间变换	cp2tform
Polynomial（多项式）	输入空间坐标是输出空间的多项式	cp2tform
Projective（投影）	如仿射变换那样，直线仍保持为直线，但平行线收敛于消失点	maketform cp2tform

例如，令 f 代表图5.15(a)所示的图像，g代表图5.15(b)所示的图像。在 f 中，控制点的坐标是(83, 81), (450, 56), (43, 293), (249, 392)和(436, 442)。在g中，相应控制点的坐标是(68, 66), (375, 47), (42, 286), (275, 434)和(523, 532)。将 g 和 f 加以对准的命令如下所示：

```
>> basepoints = [83 81; 450 56; 43 293; 249 392; 436 442];
>> inputpoints = [68 66; 375 47; 42 286; 275 434; 523 532];
>> tform = cp2tform(inputpoints, basepoints, 'projective');
>> gp = imtransform(g, tform, 'XData', [1 502], 'YData', [1 502]);
```

图5.15(c)显示了变换后的图像。

工具箱中包含一个在两幅图像上交互选择控制点的图形用户界面。图5.16显示了使用这一工具的几幅屏幕图，它由 cpselect 命令调用。



图5.16 选择控制点的交互工具

小结

本章描述了怎样用MATLAB和IPT函数进行图像复原的方法，以及怎样将其用做描述退化图像而生成模型的方法。在本章中，函数 imnoise2 和 imnoise3 的开发使得 IPT 函数对于噪声产生的能力有了显著提高。同样，函数 spfilt 可以实现的空间过滤，特别是非线性滤波，都是 IPT 函数在此领域内能力的显著扩充。这些函数同时也是将MATLAB和IPT函数联合使用的完美例子，这些新代码创造的新应用进一步增强了已有工具集的性能。

第6章　彩色图像处理

前言

在这一章中，我们将探讨利用图像处理工具箱进行彩色图像处理的基本原理，并用所开发的彩色生成和变换函数来对工具箱的功能加以扩展。本章的讨论旨在面向那些已经对彩色图像处理的基本原理和术语有一定了解的读者。

6.1　MATLAB中彩色图像的表示方法

像在2.6节中解释过的那样，图像处理工具箱将彩色图像当做索引图像或RGB图像（红、绿、蓝）来处理。在本节中，我们将详细讨论这两种类型的图像。

6.1.1　RGB图像

一幅RGB图像就是彩色像素的一个$M \times N \times 3$数组，其中每一个彩色像素点都是在特定空间位置的彩色图像相对应的红、绿、蓝三个分量（见图6.1）。RGB也可以看成是一个由三幅灰度图像形成的“堆”，当将其送到彩色监视器的红、绿、蓝输入端时，便在屏幕上产生了一幅彩色图像。按照惯例，形成一幅RGB彩色图像的三个图像常称为红、绿或蓝分量图像。分量图像的数据类决定了它们的取值范围。若一幅RGB图像的数据类是double，则它的取值范围就是[0, 1]，类似地，uint8类或uint16类RGB图像的取值范围分别是[0, 255]或[0, 65 535]。用来代表这些分量图像像素值的比特数决定了一幅RGB图像的比特深度。例如，若每个分量图像都是8比特的图像，则对应的RGB图像的深度就是24比特。一般来讲，所有分量图像的比特数都是相同的。在这种情况下，一幅RGB图像可能有的色彩数就是$(2^b)^3$，其中b是每个分量图像的比特数。对于8比特的例子，颜色数即为16 777 216。

令fR, fG和fB分别代表三种RGB分量图像。一幅RGB图像就是利用cat（级联）操作符将这些分量图像组合成的彩色图像：

```
rgb_image = cat(3, fR, fG, fB)
```

在操作中，图像按顺序放置。一般来说，cat(dim,A1,A2,...)沿着dim指定的方向级联数组。例如，若dim = 1，则数组垂直放置，若dim = 2，则数组水平放置，若dim = 3，则它们会在第三个方向放置，如图6.1所示。

若所有的分量图像都是一样的，则结果是一幅灰度图像。令rgb_image代表一幅RGB图像。下面的命令可以提取出三幅分量图像：

```
>> fR = rgb_image(:, :, 1);
>> fG = rgb_image(:, :, 2);
>> fB = rgb_image(:, :, 3);
```

RGB彩色空间常常用一个RGB彩色立方体加以图解展示，如图6.2所示。这个立方体的顶点是光的原色（红、绿、蓝）和合成色（青、品红、黄）。

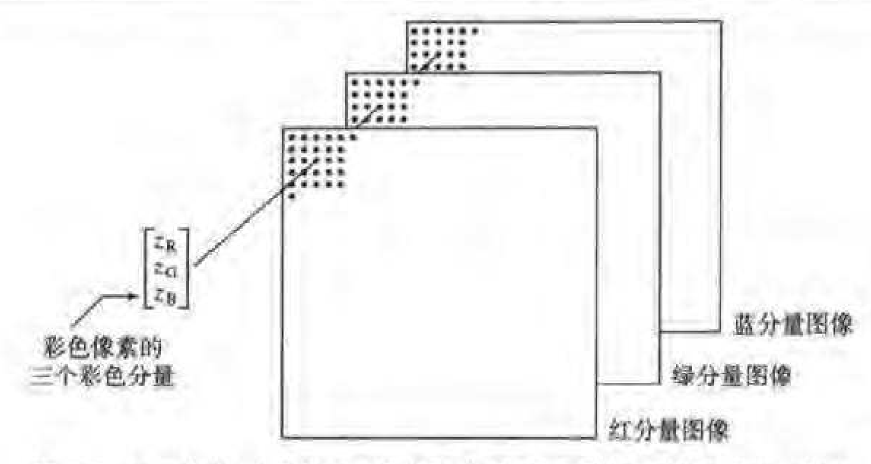


图 6.1　由三个分量图像的相应像素形成 RGB 彩色图像像素的示意图

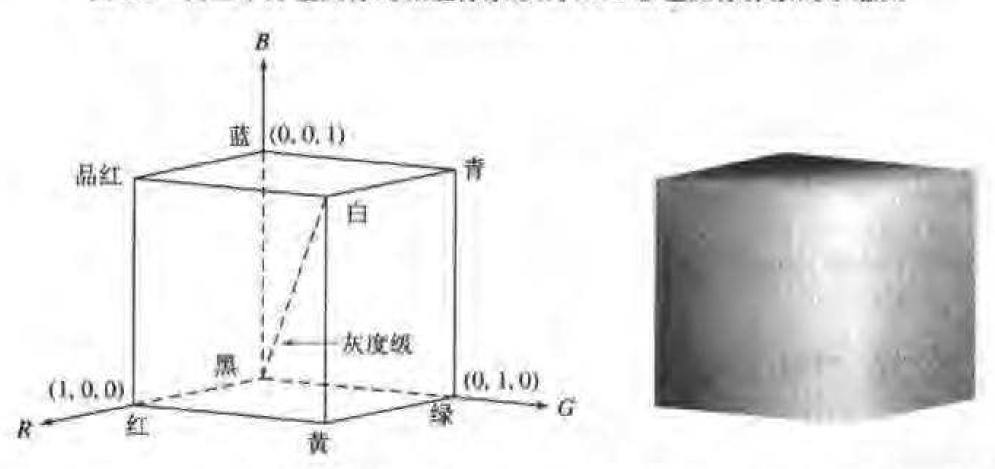


图 6.2　(a)在原点显示光的原色和合成色的RGB彩色立方体示意图。沿主对角线的点有从原点的黑色到点（1, 1, 1）的白色的灰度值；(b)RGB 彩色立方体

通常，能够从任何透视方向观察这个彩色立方体是很有用的。函数 rgbcube 就用于这一目的其语法为

```
rgbcube( vx, vy, vz)
```

在提示符处键入rgbcube(vx,vy,vz)便会在MATLAB桌面上生成一个从点(vx,vy,vz)观察的 RGB 立方体。结果图像可如 2.4 节讨论过的那样利用函数 print 存储到磁盘上。该函数的代码如下所示。

```
function rgbcube(vx, vy, vz)
%RGBCUBE Displays an RGB cube on the MATLAB desktop.
%   RGBCUBE(VX, VY, VZ) displays an RGB color cube, viewed from point
%   (VX, VY, VZ). With no input arguments, RGBCUBE uses (10,10,4)
%   as the default viewing coordinates. To view individual color
%   planes, use the following viewing coordinates, where the first
%   color in the sequence is the closest to the viewing axis, and the
%   other colors are as seen from that axis, proceeding to the right
%   right (or above), and then moving clockwise.
%
%       ---------------------------------------------------
%           COLOR PLANE                      ( vx,   vy,   vz)
%       ---------------------------------------------------
%         Blue-Magenta-White-Cyan            (  0,    0,   10)
```

```
%       Red-Yellow-White-Magenta         ( 10,    0,    0)
%       Green-Cyan-White-Yellow          (  0,   10,    0)
%       Black-Red-Magenta-Blue           (  0,  -10,    0)
%       Black-Blue-Cyan-Green            (-10,    0,    0)
%       Black-Red-Yellow-Green           (  0,    0,  -10)

% Set up parameters for function patch.
vertices_matrix = [0 0 0;0 0 1;0 1 0;0 1 1;1 0 0;1 0 1;1 1 0;1 1 1];
faces_matrix = [1 5 6 2;1 3 7 5;1 2 4 3;2 4 8 6;3 7 8 4;5 6 8 7];
colors = vertices_matrix;
% The order of the cube vertices was selected to be the same as
% the  order of the (R,G,B) colors (e.g., (0,0,0)corresponds to
% black, (1 1 1) corresponds to white, and so on.)

% Generate RGB cube using function patch.
patch('Vertices', vertices_matrix, 'Faces', faces_matrix, ...
      'FaceVertexCData', colors, 'FaceColor', 'interp', ...
      'EdgeAlpha', 0)

% Set up viewing point.
if nargin == 0
   vx = 10; vy = 10; vz = 4;
elseif nargin ~= 3
   error('Wrong number of inputs.')
end
axis off
view([vx, vy, vz])
axis square
```

6.1.2 索引图像

索引图像有两个分量，即整数的数据矩阵X和彩色映射矩阵map。矩阵map是一个大小为$m\times3$且由范围在[0, 1]之间的浮点值构成的double类数组。map的长度m同它所定义的颜色数目相等。map的每一行都定义单色的红、绿、蓝三个分量。索引图像将像素的亮度值"直接映射"到彩色值。每个像素的颜色由对应的整数矩阵X的值作为指向map的一个指针决定。若X属double类，则其小于或等于1的所有分量都指向map的第1行，所有等于2的分量都指向第2行，依次类推。若X为uint8类或uint16类图像，则所有等于零的分量都指向map的第1行，所有等于1的分量都指向第2行，依次类推。这些概念都在图6.3中给予说明。

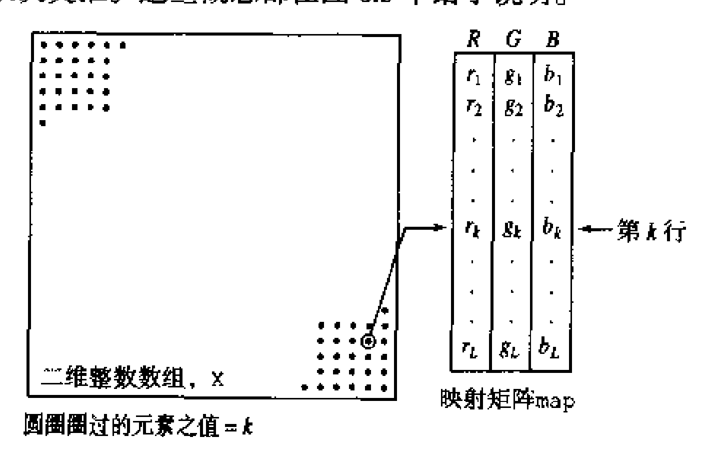


图6.3 索引图像的元素。注意，整数数组X的元素的值决定彩色映射的行数。每一行包含一个RGB三元组，L是总行数

为了显示出一幅索引图像，可使用语句

```
>> imshow(X, map)
```

或语句

```
>> image(X)
>> colormap(map)
```

colormap是和索引图像共同存储的，当用函数imread加载图像时，它会自动地和图像一起载入。

有时需要用较少的颜色来近似一幅索引图像。为此，我们引入函数imapprox，其语法为

```
[Y, newmap] = imapprox(X, map, n)
```

该函数利用彩色映射newmap返回一个数组Y，该数组最多有n种颜色。输入数组X可以是uint8类、uint16类或double类。若n小于或等于256，则输出数组Y是uint8类，若n大于256，则Y是double类。

当map中的行数比X中的不同整数值的数目少时，X中的多个值将在map中用同样的颜色加以显示。例如，假设X由4个垂直等宽度区域组成，它们的值分别为1, 64, 128和256。若指定彩色映射map=[0 0 0;1 1 1]，则所有X中的值为1的元素就会指向该图的第一行（黑色），其他所有的元素都将指向第二行（白色）。从而，命令imshow(X,map)的执行会显示出一幅由一个黑色区域后面紧跟三个白色区域的图像。事实上，只要图的长度是65，这都是准确的。当长度为65时，会显示一个黑色区域，后面紧跟着一个灰色区域，然后是两个白色区域。若图的长度超过了X中的元素的允许值的范围，则就会得出无意义的图像。

指定一幅彩色图的方法有很多。一种方法就是利用如下语句：

```
>> map(k, :) = [r(k) g(k) b(k)]
```

其中[r(k) g(k) b(k)]是RGB值，它指定彩色映射的一行。变化k的值就可将图填满。

表6.1列出了一些基本颜色的RGB值。表中的三种格式都可以单独用来指定颜色。例如，要想把一幅图像的背景色改成绿色，可用下面的三个语句之一来完成：

```
>> whitebg('g')
>> whitebg('green')
>> whitebg([0 1 0])
```

表6.1 一些基本颜色的RGB值。可以用长名或短名（用引号括起来）代替数值三元组来指定一套RGB颜色

长名	短名	RGB值
Black（黑）	k	[0 0 0]
Blue（蓝）	b	[0 0 1]
Green（绿）	g	[0 1 0]
Cyan（青）	c	[0 1 1]
Red（红）	r	[1 0 0]
Magenta（品红）	m	[1 0 1]
Yellow（黄）	y	[1 1 0]
White（白）	w	[1 1 1]

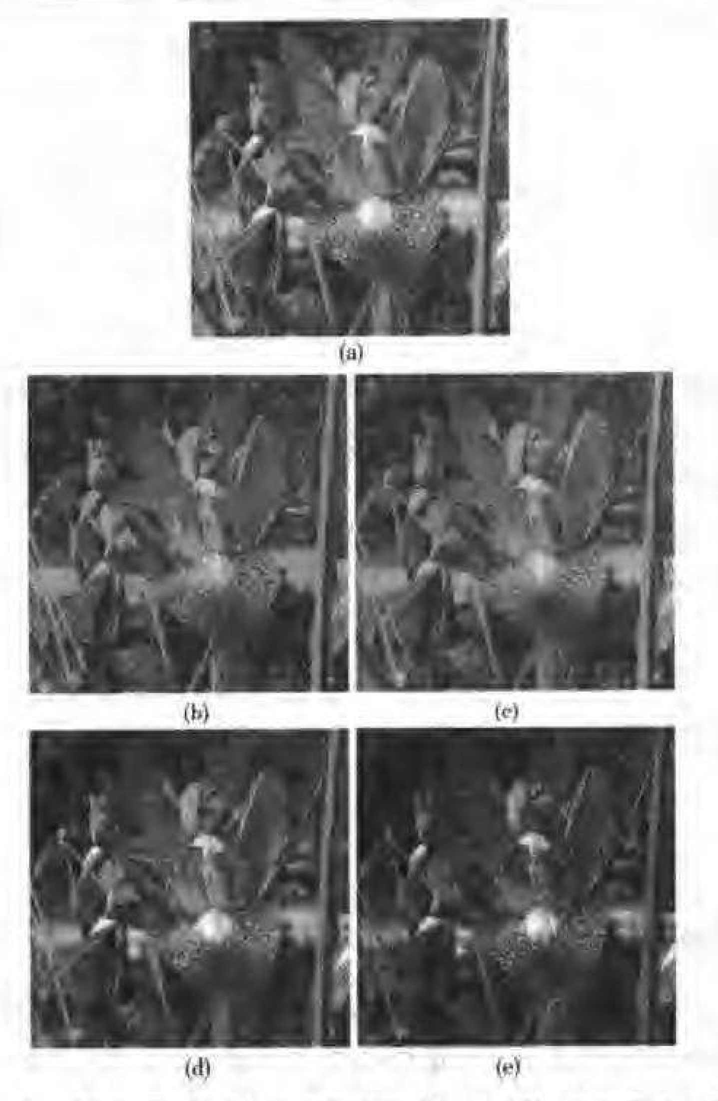


图6.4　(a)RGB图像；(b)无抖动处理的颜色数减少到8的图像；(c)有抖动处理的颜色数减少到8的图像；(d)使用函数 rgb2gray 得到的(a)的灰度级图像；(e)经抖动处理后的灰度图像（这是一幅二值图像）

6.2　转换至其他彩色空间

如在前节所解释的那样，工具箱用RGB值在RGB图像中直接描述颜色，或者在索引图像中间接描述颜色，此时，彩色映射使用RGB格式来存储。然而，还有其他的彩色空间（又称为彩色模型），它们的应用有时会更加方便或更加恰当。其中包括NTSC、YCbCr、HSV、CMY、CMYK和HSI彩色空间。工具箱提供了可以由RGB向NTSC、YCbCr、HSV、CMY转换的函数，反之亦然。将彩色空间转换成HSI彩色空间并可转回的函数将在本节后面讨论。

6.2.1　NTSC彩色空间

NTSC彩色制式在美国用于电视系统。这种形式的一个主要优势是灰度信息和彩色信息是分离的，所以同一个信号既可以用于彩色电视机，又可用于黑白电视机。在NTSC制式中，图像数据是由三部分组成的：亮度（Y）、色调（I）和饱和度（Q），其中字母YIQ的选择常常是按照惯例进行的。亮度分量描述灰度信息，其他两个分量携带电视信号的彩色信息。YIQ这几个分量都是利用如下变换从一幅图像的RGB分量中得到的：

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

注意，第一行的各元素之和为1，而下两行的和为0。这和预想的一样，因为在一幅灰度图像中，所有的RGB分量都相等，所以对这样的图像来说，I和Q分量必然为零。函数 rgb2ntsc 可执行这样的变换：

```
yiq_image = rgb2ntsc(rgb_image)
```

其中，输入RGB图像可以是uint8类、uint16类或double类。输出图像为double类图像，大小为$M \times N \times 3$。分量图像 yiq_image(:,:,1)代表亮度，yiq_image(:,:,2)代表色度，yiq_image(:,:,3)代表饱和度。

类似地，RGB分量可以利用下面的变换从YIQ分量获得：

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.956 & 0.621 \\ 1.000 & -0.272 & -0.647 \\ 1.000 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

IPT 函数 ntsc2rgb 用于实现该公式：

```
rgb_image = ntsc2rgb(yiq_image)
```

输入和输出图像都是 double 类图像。

6.2.2 YCbCr 彩色空间

YCbCr彩色空间广泛应用于数字视频。在这种格式中，亮度信息用单个分量Y来表示，彩色信息用两个色差分量Cb和Cr来存储。分量Cb是蓝色分量和一个参考值的差，分量Cr是红色分量和一个参考值的差（见Poynton[1996]）。IPT将RGB转换为YCbCr所用的变换是

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 65.481 & 128.553 & 24.966 \\ -37.797 & -74.203 & 112.000 \\ 112.000 & -93.786 & -18.214 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

转换函数是

```
ycbcr_image = rgb2ycbcr(rgb_image)
```

输入RGB图像可以是uint8类、uint16类或double类。输出图像和输入图像有着相同的类。类似的变换可以将YCbCr转换为RGB：

```
rgb_image = ycbCr2rgb(ycbcr_image)
```

输入YCbCr图像可以是uint8类、uint16类或double类。输出图像和输入图像有着相同的类。

6.2.3 HSV 彩色空间

HSV（色调、饱和度、数值）是人们用来从调色板或颜色轮中挑选颜色（如颜料或墨水）所用的彩色系统之一。值得注意的是，该颜色系统比RGB系统更接近于人们的经验和对彩色的感知。在画家的术语里，色调、饱和度和数值称为色泽、明暗和调色。

HSV彩色空间可以通过沿灰度轴（连接黑色顶点和白色顶点的轴）来研究RGB彩色立方体加以公式化表达，从而得出图6.5(a)所示的六边形彩色调色板。当沿如图6.5(b)中的垂直（灰度）轴移动时，这个六边形平面的大小是变化的，平面与轴垂直，并产生了图中所示的锥体。

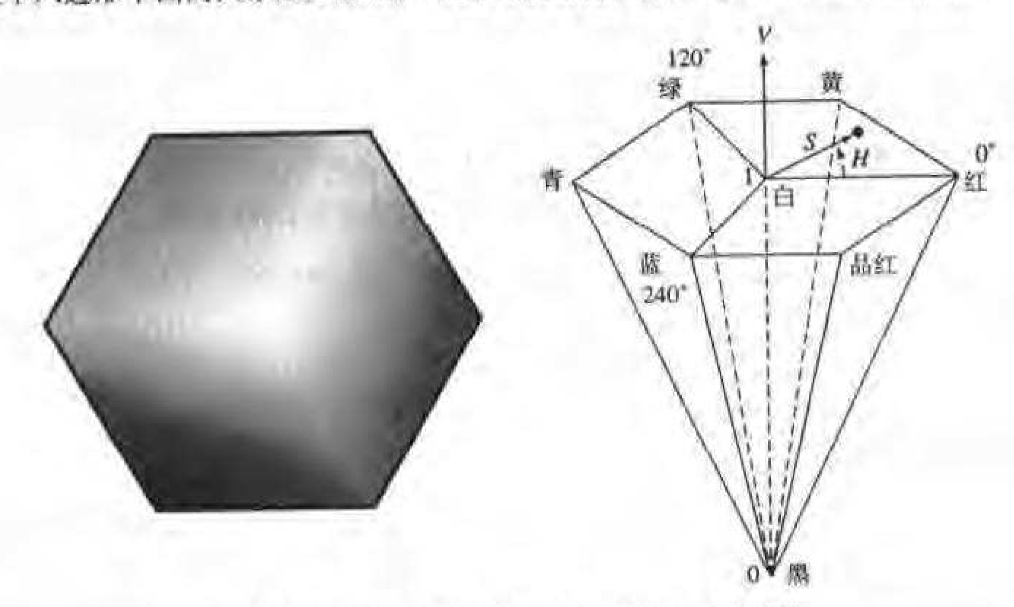


图6.5 (a)HSV彩色六边形；(b)HSV六面锥体

色调是用围绕彩色六边形的角度来描述的，一般使用红色轴作为0°轴。沿锥体的轴来测量值。当$V=0$时，轴的末端为黑色；当$V=1$时，轴的末端为白色，这些轴位于图6.5(a)所示的全彩六边形中。从而，该轴代表了所有灰度级。饱和度（颜色的纯净度）是指到V轴的距离。

HSV彩色系统基于柱坐标系。将RGB转换为HSV就是简单地将方程式展开，从而将RGB值（笛卡儿坐标系）映射至柱坐标系。大多数计算机图形学的文献都对这一问题给予了详细论述（见Rogers[1997]），所以我们在此不再展开这个方程式。

将RGB转换为HSV的MATLAB函数是rgb2hsv，其语法为

```
hsv_image = rgb2hsv(rgb_image)
```

输入RGB图像可以是uint8类、uint16类或者double类，输出图像是double类。将HSV转换为RGB的函数是hsv2rgb，其语法为

```
rgb_image = hsv2rgb(hsv_image)
```

输入图像必须是double类，输出图像也是double类。

6.2.4 CMY和CMYK彩色空间

青色、品红色和黄色是光的合成色，换言之，是颜料的原色。例如，当在一个表面涂上青色颜料并使用白光加以照射时，表面不会反射红光。因为青色颜料从反射的白光中吸收了红光，而白光本身是由等量的红光、绿光和蓝光组成的。

大多数将颜料沉淀于纸上的设备，如彩色打印机和复印机，都需要CMY数据输入，或在内部将RGB转换为CMY。该转换用到了方程

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

其中，假设所有的颜色值都已归一化到范围[0, 1]。该方程证明了从一个涂满纯净青色颜料的表面反射回的光不包含红色（即方程中 $C = 1 - R$）。同样，纯净的品红色不反射绿色，纯净的黄色不反射蓝色。前述的方程同样表明，从1减去单个CMY值，可以从CMY值的集合中获得RGB值。

理论上，等量的颜料原色即青色、品红色和黄色混合会产生黑色。在实践中，将这些颜色加以混合来印刷会生成一幅模糊不清的黑色图像。所以，为了生成一种纯正的黑色（打印中使用的主要颜色），便要添加第四种颜色，即黑色，从而出现了CMYK彩色模型。这样，当出版者谈论“四色印刷”时，他们其实是在说CMY彩色模型的三种颜色加上黑色。

3.2.1节中介绍的 `imcomplement` 函数可把RGB图像转换为CMY图像：

```
cmy_image = imcomplement(rgb_image)
```

也可以用该函数将CMY图像转换为RGB图像：

```
rgb_image = imcomplement(cmy_image)
```

6.2.5 HSI彩色空间

除HSV外，至今为止所讨论过的彩色空间还没有适合人们以术语描述彩色的方式。例如，一个人想描述汽车的颜色时，不可能用给出构成该颜色的颜料原色中各个分量的百分比这种办法。

当人们观察一个彩色的物体时，往往倾向于用它的色调、饱和度和明度来描述它。饱和度给出了纯彩色被白光冲淡程度的度量，而色调描述纯色的属性（如纯黄色、橙色或红色）。明度是一个主观描述符，事实上几乎无法度量。它使非彩色的亮度概念得以具体化，并且是描述颜色感觉的一个关键因素。我们已经知道，亮度（灰度级）是描述单色图像的一个最有用的描述符。这个明确的量可以被度量，也可以很容易地描述出来。

我们将要讨论的彩色空间称为HSI（hue，色度；saturation，饱和度；intensity，亮度）彩色空间，该模型将亮度分量与一幅彩色图像中携带的彩色信息分开。因此，HSI模型对于开发基于彩色描述的图像处理算法是一个理想的工具，对于人类来说，它们看起来更加自然和直观，毕竟，人是这些算法的开发者和使用者。HSV彩色空间与此类似，但它侧重于展示出一些有意义的色彩，就像对一名画家所用的调色板的解释那样。

如在6.1.1节中讨论过的那样，RGB彩色图像是由三幅单色的亮度图像构成的，所以一定可以从一幅RGB图像中提取出亮度。若拿图6.2所示的彩色立方体来看，一切就很清楚。假设我们站在黑色顶点(0, 0, 0)处，如图6.6(a)所示，它的正上方是白色顶点（1, 1, 1）。再同图6.2联系起来看，亮度是沿着连接着两个点的连线分布的。在图6.6所示的排列中，这条连接黑色和白色顶点的线（亮度轴）是垂直的。因此，若想确定图6.6中任意彩色点的亮度分量，我们就需要经过一个包含该点且垂直于亮度轴的平面。这个平面和亮度轴的交点就给出了范围在[0, 1]之间的亮度值。我们也注意到饱和度是一个与亮度轴之间的距离的函数。事实上，亮度轴上的点的饱和度为零，亮度轴上的所有点都是灰色的这个事实是很显然的。

为了弄清从一个给定RGB点确定色调的方式，可参考图6.6(b)，它显示了一个由三个点（黑色、白色和青色）所定义的平面。该平面上含有黑色和白色顶点的事实告诉我们亮度轴同样在这个平面上。此外，可以看到由亮度轴和立方体边界共同定义的平面上的所有点都有相同的色调（在此例中为青色）。这是因为在一个彩色三角形内，颜色是由这三个顶点颜色的多种多样的组合或者混合而成的。若这些顶点中的两个是黑色和白色，第三个顶点是一个彩色的点，则这个三角形中所有点的色调都是相同的，因为白色和黑色分量对于色彩的变化没有影响（当然，在这个三角形中，点的亮度和饱和度会变化）。以垂直的强度轴旋转这个阴影平面，可以获得不同的色调。从这些概念

出发，我们得到如下结论：形成一个 HSI 空间所需的色调、饱和度和亮度值可以通过 RGB 彩色立方体得到。通过算出刚刚在前边概略推出的几何公式，就可以将任意的 RGB 点转换成 HSI 彩色模型中的对应点。

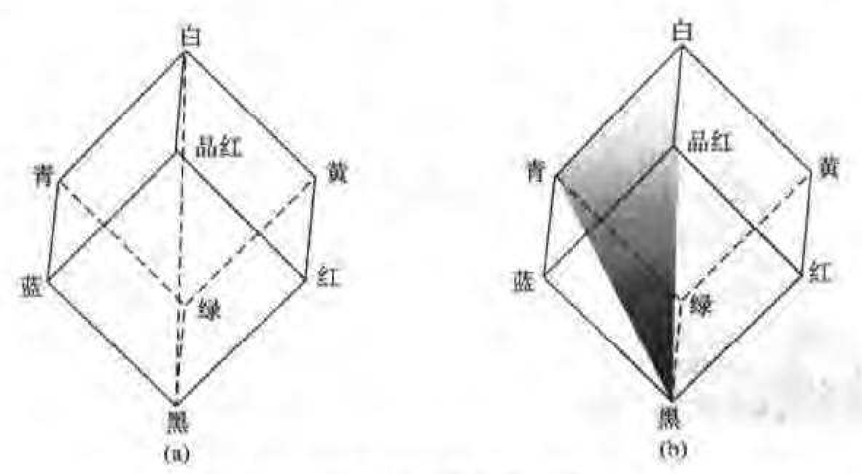


图 6.6　RGB 与 HSI 彩色模型间的关系

基于前面的讨论，我们认识到，HSI 空间由一个垂直的亮度轴以及垂直于此轴的一个平面上的彩色点的轨迹组成。随着平面在亮度轴上下移动，由平面和立方体表面相交形成的边界或者是三角形或者是六边形。若从立方体的灰度轴看去，如图 6.7(a)所示，这个结论会变得更加直观。在这个平面上，我们看到各原色之间都相隔了 120°。合成色与原色相关 60°，这意味着合成色间的角度也是 120°。

图 6.7(b)显示了六边形和一个任意的彩色点（用点的形式显示）。这个点的色调是由其与某参考点的夹角决定的。一般来说（但也不总是如此），与红色轴的夹角为 0°，色调在此处按逆时针方向增加。饱和度（距垂直轴的距离）是从原点到该点的向量的长度。注意，原点由该彩色平面与垂直亮度轴的交点定义。HSI 彩色空间的重要分量包括垂直亮度轴、到彩色点的向量长度以及该向量与红色轴间的夹角。因此，当用刚才讨论过的六边形甚至如图 6.7(c)和如图 6.7(d)所示的三角形或圆形时，这些形式定义一个 HSI 平面是不足为奇的。选择哪种形状是不重要的，因为任意一种形状都可以通过几何变换变换成另一种形状。图 6.8 显示了基于彩色三角形和圆形的 HSI 模型。

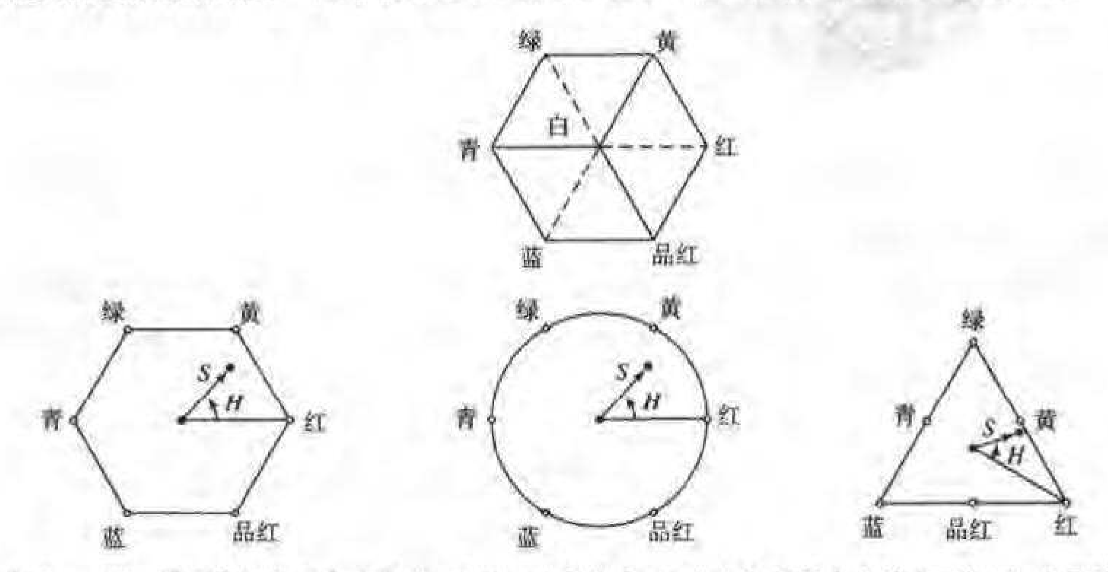


图 6.7　HSI 模型中的色调和饱和度。点是一个任意点，它与红色轴的夹角给出了色调，向量的长度是饱和度。在这些平面中的所有彩色的亮度都由垂直亮度轴上的平面的位置给出

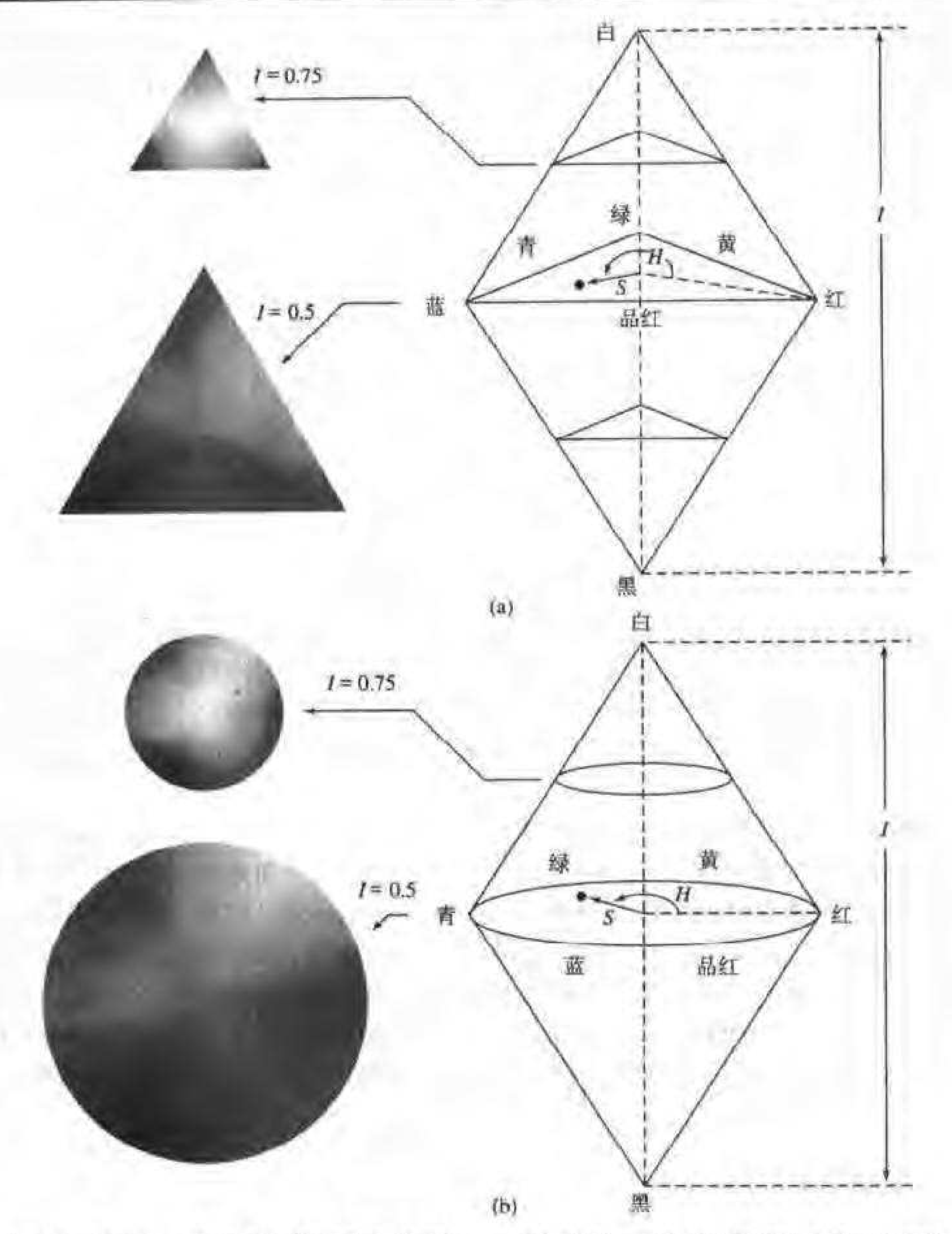


图 6.8　基于(a)三角形和(b)圆形彩色平面的 HSI 彩色模型。三角形和圆形均垂直于亮度轴

将颜色从 RGB 转换为 HSI

在下面的讨论中，我们给出从 RGB 到 HSI 的转换方程，但不予推导。详细推导过程可参考本书的网站（1.5 节中列出了地址）。若给出一幅 RGB 彩色格式的图像，则每一个 RGB 像素的 H 分量可用下面的方程得出：

$$H = \begin{cases} \theta & \text{若 } B \leqslant G \\ 360 - \theta & \text{若 } B > G \end{cases}$$

其中，

$$\theta = \arccos\left\{\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{1/2}}\right\}$$

饱和度分量由下式给出：

$$S = 1 - \frac{3}{(R+G+B)}[\min(R,G,B)]$$

最后，亮度由下式给出：

$$I = \frac{1}{3}(R+G+B)$$

假定RGB值已归一化到范围[0, 1]，角度θ是点与HSI空间的红色轴之间的夹角，如图6.7所示。将由H的公式中得出的所有结果值除以360°，色度可归化到范围[0, 1]。若给出的RGB值在[0, 1]之间，则剩下的两个HSI分量自然就会在[0, 1]之间。

将颜色从HSI转换为RGB

给出区间[0, 1]内的HSI值后，我们现在找出同一区间内对应的RGB值。可用的公式依赖于H的值。有三个感兴趣的区域，分别对应于原色之间相隔120°的区间（见图6.7）。让H乘以360°，就可将色调的值还原到其原来的范围[0°, 360°]。

RG区（$0° \leqslant H < 120°$）：若H在这个区域内，则RGB分量由式

$$B = I(1-S)$$

$$R = I\left[1 + \frac{S\cos H}{\cos(60° - H)}\right]$$

和

$$G = 3I - (R+B)$$

给出。

GB区（$120° \leqslant H < 240°$）：若H的给定值在这个区域内，我们就先从中减去120°：

$$H = H - 120°$$

则RGB分量为

$$R = I(1-S)$$

$$G = I\left[1 + \frac{S\cos H}{\cos(60° - H)}\right]$$

和

$$B = 3I - (R+G)$$

BR区（$240° \leqslant H \leqslant 360°$）：最后，若$H$在这个区域内，我们就从中减去240°：

$$H = H - 240°$$

则RGB分量分为

$$G = I(1-S)$$

$$B = I\left[1 + \frac{S\cos H}{\cos(60° - H)}\right]$$

和

$$R = 3I - (G+B)$$

这些公式在图像处理中的应用将在本章稍后讨论。

从 RGB 转换到 HSI 的一个 M 函数

函数

```
hsi = rgb2hsi(rgb)
```

用于实现刚刚讨论过的把 RGB 转换成 HSI 格式的公式。为简化符号，我们分别用 rgb 和 hsi 来表示 RGB 和 HSI 图像。代码中的文本详述了该函数的用途。

```
function hsi = rgb2hsi(rgb)
%RGB2HSI Converts an RGB image to HSI.
%   HSI = RGB2HSI(RGB) converts an RGB image to HSI. The input image
%   is assumed to be of size M-by-N-by-3, where the third dimension
%   accounts for three image planes: red, green, and blue, in that
%   order. If all RGB component images are equal, the HSI conversion
%   is undefined. The input image can be of class double (with values
%   in the range [0, 1]), uint8, or uint16.
%
%   The output image, is of class double, where:
%      hsi(:, :, 1) = hue image normalized to the range [0,1] by
%                     dividing all angle values by 2*pi.
%      hsi(:, :, 2) = saturation image, in the range [0, 1].
%      hsi(:, :, 3) = intensity image, in the range [0, 1].

% Extract the individual component immages.
rgb = im2double(rgb);
r = rgb(:, :, 1);
g = rgb(:, :, 2);
b = rgb(:, :, 3);

% Implement the conversion equations.
num = 0.5*((r - g) + (r - b));
den = sqrt((r - g).^2 + (r - b).*(g - b));
theta = acos(num./(den + eps));

H = theta;
H(b > g) = 2*pi - H(b > g);
H = H/(2*pi);

num = min(min(r, g), b);
den = r + g + b;
den(den == 0) = eps;
S = 1 - 3.* num./den;

H(S == 0) = 0;

I = (r + g + b)/3;

% Combine all three results into an hsi image.
hsi = cat(3, H, S, I);
```

从 HSI 转换到 RGB 的一个 M 函数

函数

```
rgb = hsi2rgb(hsi)
```

用于实现把 HSI 转换成 RGB 的公式。代码中的文本详述了该函数的用途。

```
function rgb = hsi2rgb(hsi)
%HSI2RGB Converts an HSI image to RGB.
```

```
%   RGB = HSI2RGB(HSI) converst an HSI image to RGB, where HSI is
%   assumed to be of class double with:
%      hsi(:, :, 1) = hue image, assumed to be in the range
%                     [0, 1] by having been divided by 2*pi.
%      hsi(:, :, 2) = saturation image, in the range [0, 1].
%      hsi(:, :, 3) = intensity image, in the range [0, 1].
%
%   The components of the output image are:
%      rgb(:, :, 1) = red.
%      rgb(:, :, 2) = green.
%      rgb(:, :, 3) = blue.

% Extract the individual HSI component images.
H = hsi(:, :, 1) * 2 * pi;
S = hsi(:, :, 2);
I = hsi(:, :, 3);

% Implement the conversion equations.
R = zeros(size(hsi, 1), size(hsi, 2));
G = zeros(size(hsi, 1), size(hsi, 2));
B = zeros(size(hsi, 1), size(hsi, 2));

% RG sector (0 <= H < 2*pi/3).
idx = find( (0 <= H) & (H < 2*pi/3));
B(idx) = I(idx) .* (1 - S(idx));
R(idx) = I(idx) .* (1 + S(idx) .* cos(H(idx))./ ...
                                          cos(pi/3 - H(idx)));
G(idx) = 3*I(idx) - (R(idx) + B(idx));

% BG sector (2*pi/3 <= H < 4*pi/3).
idx = find( (2*pi/3 <= H) & (H < 4*pi/3) );
R(idx) = I(idx) .* (1 - S(idx));
G(idx) = I(idx) .* (1 + S(idx) .* cos(H(idx) - 2*pi/3) ./ ...
                    cos (pi - H(idx)));
B(idx) = 3*I(idx) - (R(idx) + G(idx));

% BR sector.
idx = find( (4*pi/3 <= H) & (H <= 2*pi));
G(idx) = I(idx).* (1 - S(idx));
B(idx) = I(idx).* (1 + S(idx).* cos(H(idx) - 4*pi/3)./ ...
                                          cos(5*pi/3 - H(idx)));
R(idx) = 3*I(idx) - (G(idx) + B(idx));

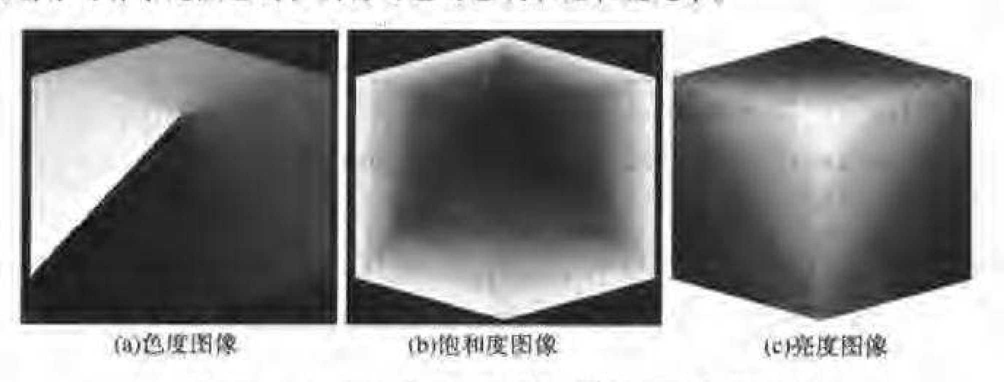
% Combine all three results into an RGB image. Clip to [0, 1] to
% compensate for floating-point arithmetic rounding effects.
rgb = cat(3, R, G, B);
rgb = max(min(rgb, 1), 0);
```

例6.2　从RGB转换到HSI

图6.9显示了白色背景下一个RGB立方体的图像的色调、饱和度和亮度分量，这类似于图6.2(b)所示的图像。图6.9(a)是色调图像。它的可区别特征是在立方体的前（红色）平面中沿45°线的值的不连续。为了解这种不连续的原因，可参考图6.2(b)，从立方体的红顶点到白顶点画一条线，并在这条线的中间选择一个点。从该点开始向右画一条路径，环绕着立方体直到返回起始点。交汇到这一通路上的颜色是黄、绿、青、蓝、品红、黑和红。根据图6.7，色调的值沿

着这条通路应该从0°到360°增加（即从色调的最低值到最高值）。而这正是图6.9(a)所示的内容，因为最低值表示黑色，最高值表示白色。

图6.9(b)中的饱和度图像显示了较暗的值到RGB立方体的白色顶点渐渐增加，表明彩色饱和度越来越弱直至白色。最后，图6.9(c)显示的亮度图像中的每个像素都是相应于图6.2(b)中RGB像素值的平均值。注意，这幅图像的背景是白色，因为背景的亮度在彩色图像中是白色的。其他两幅图像的背景是黑色的，因为白色的色调和饱和度是零。



(a)色度图像　　(b)饱和度图像　　(c)亮度图像

图6.9　一个RGB彩色立方体的图像的HSI分量图像

6.3　彩色图像处理基础

在这一节中，我们开始研究用于彩色图像的处理技术。虽然未给出所有的技术，但本节给出的技术说明了各种图像处理任务处理彩色图像的方法。为了下面讨论的目的，我们把彩色图像处理细分成三个主要类别：(1) 颜色变换（也称为彩色映射）；(2) 单独彩色平面的空间处理；(3) 颜色向量处理。

第一类处理每个彩色平面的像素，该处理严格地以像素值为基础，而不是以它们的空间坐标为基础。这类处理类似于3.2节中的亮度变换处理。第二类对各个彩色平面进行空间（邻域）滤波，类似于3.4节和3.5节中讨论的空间滤波。第三类是以同时处理彩色图像的所有分量为基础的处理技术。因为全彩色图像至少有三个分量，彩色像素实际上是向量。例如，在RGB系统中，每个彩色点都会在RGB坐标系中作为一个从原点延伸到那一点的向量来描述（见图6.2）。

令$\mathbf{c}$代表RGB彩色空间中的任意向量：

$$\mathbf{c} = \begin{bmatrix} c_R \\ c_G \\ c_B \end{bmatrix} = \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

该公式表明$\mathbf{c}$的分量是一幅彩色图像在一个点上的RGB分量。彩色分量是坐标(x, y)的函数，表示为

$$\mathbf{c}(x, y) = \begin{bmatrix} c_R(x, y) \\ c_G(x, y) \\ c_B(x, y) \end{bmatrix} = \begin{bmatrix} R(x, y) \\ G(x, y) \\ B(x, y) \end{bmatrix}$$

对一个大小为$M \times N$的图像来说，有MN个这样的向量$\mathbf{c}(x, y)$，其中，$x = 0, 1, 2, \cdots, M-1$和$y = 0, 1, 2, \cdots, N-1$。

在某些情况下，无论是一次处理彩色图像的一个平面，还是作为向量来处理，都会得到相同的结果。然而，如我们将在6.6节中详细描述的那样，它并不总是如此。为了使独立的彩色分量和以向量为基础的处理都相同，必须满足两个条件：首先，该处理必须对向量和标量都可用；其次，对向量的每个分量的运算必须独立于其他分量。例如，图6.10所示的灰度级及全彩色图像的空间邻域

它返回一个列向量，该向量包括在点 xi 处使用一维函数 z 线性插值的值。列向量 x 和 y 指定控制点的水平和垂直坐标。x 的元素必须单调增长。z 的长度等于 xi 的长度。例如，

```
>> z = interp1q([0 255]', [0 255]', [0: 255]')
```

产生一个有着256个元素的一一映射，以连接控制点（0,0）和（255,255），即 z=[0 1 2...255]'。

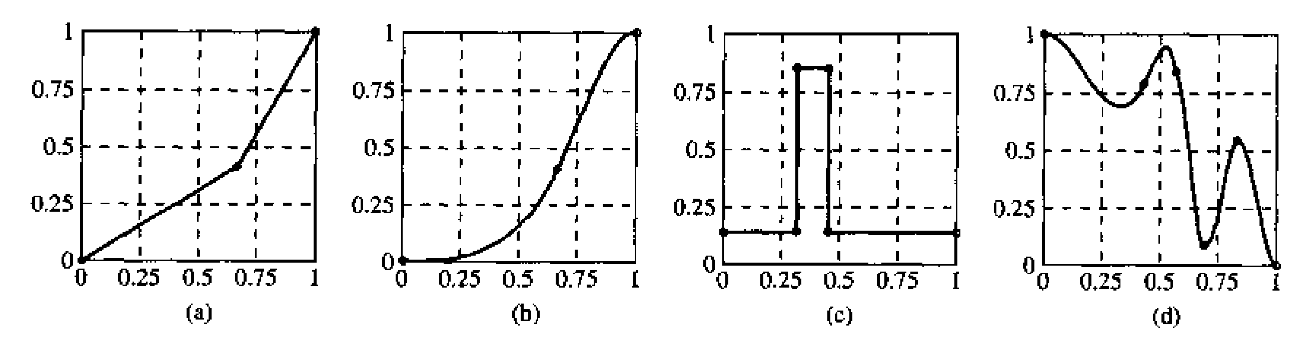


图6.11 指定使用控制点的映射函数：(a)和(c)线性插值；(b)和(d)三次样条插值

类似地，三次样条插值使用 spline 函数实现：

```
z = spline(x, y, xi)
```

其中，变量 x, y, z 和 xi 已在前一段描述 interp1q 时做了说明。然而，xi 不同于函数 spline。此外，若 y 比 x 多包含两个元素，则其第一个和最后一个输入被假定为三次样条的端滚降。例如，图6.11(b)中描述的函数是使用零值端滚降产生的。

变换函数的说明可以按图形方式操作控制点来交互地产生，这些控制点是函数 interp1q 和 spline 的输入并实时地在被处理的图像上显示变换函数的结果。ice（交互彩色编辑）函数可准确地做到这一点，其语法为

```
g = ice('Property Name', 'Property Value', . . .)
```

其中，'Property Name' 和 'Property Value' 必须成对出现，并且这些点表示由相应输入对所组成的模式的重复。表6.4列出了 ice 函数中的正确搭配。一些例子将在本章稍后给出。

关于 'wait' 参数，当显式地或默认地选择 'on' 时，输出 g 是处理后的图像。在这种情况下，ice 将控制处理，包括光标，因而在命令窗口不必键入任何命令，直到函数关闭，这时最后的结果就是图像 g。当选择 'off' 时，g 为处理后的图像的句柄[①]，并且控制会立即返回到命令窗口；因此，在函数 ice 仍处于活动状态时，可以键入新的命令。为了用句柄 g 获得图像的属性，我们使用 get 函数

```
h = get(g)
```

表6.4 函数 ice 的有效输入

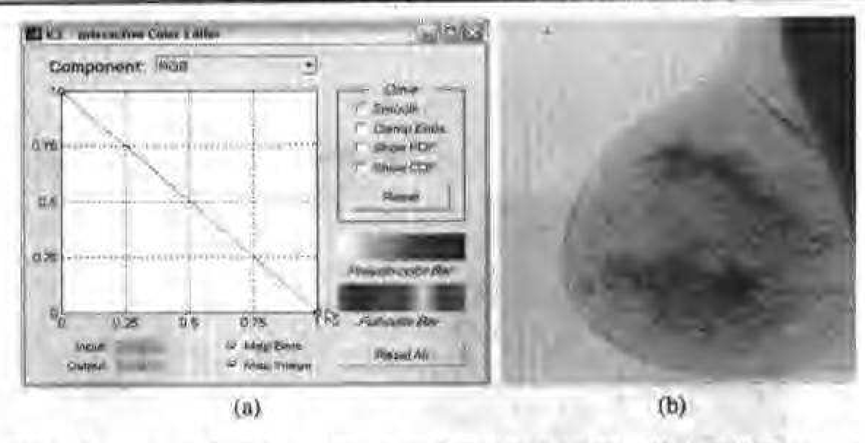
属性名称	属性值
'image'	一幅 RGB 或单色输入图像 f，它由指定的映射交互式地变换
'space'	将被修改的分量的彩色空间。可能的值是 'rgb', 'cmy', 'hsi', 'hsv', 'ntsc'（或 'yiq'）和 'ycbcr'。默认值为 'rgb'
'wait'	若值为 'on'（默认），则 g 是被映射的输入图像。若值为 'off'，则 g 是被映射的输入图像的句柄

① MATLAB 创建一个图形对象后，会给该对象分配一个身份标识（称为句柄），以用于访问该对象的属性。在修改图形的外观或通过编写直接创建和操纵对象的 M 文件而创建自定义绘图命令时，图形句柄是很有用的。

的方式。仅修改了RGB映射；R，G和B映射分别保留了默认的1:1状态（见表6.6中的Component部分）。对于单色输入，这保证了单色输出，图6.13(b)显示了逆映射导致的单色负片。注意，它与图3.3(b)完全相同，而后者是使用函数 `imcomplement` 得到的。图6.13(a)中的伪彩色条是图6.12中的原始灰度条的"负片"。

表6.6 函数 `ice` 的图形用户界面中的按扭和复选框的功能

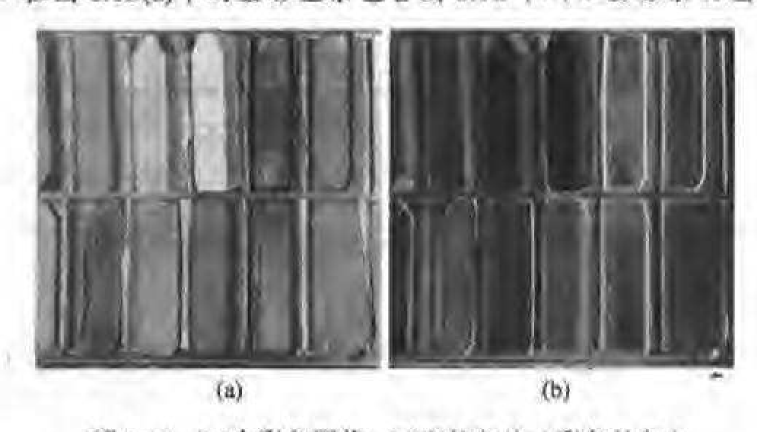
GUI元素	功能
Smooth	若选中，则进行三次样条（平滑曲线）插值。若不选，则分段地进行线性插值
Clamp Ends	若选中，则在三次样条插值中强制开始和结束曲线斜降为零。分段地进行线性插值不受影响
Show PDF	显示受映射函数影响的图像分量的概率密度函数（如直方图）
Show CDF	显示累积分布函数而不是PDF（注意，PDF和CDF不能同时显示）
Map Image	若选中，则启用图像映射；否则不启用图像映射
Map Bars	若选中，则启用伪彩色和全彩色条映射；否则，显示非映射条（灰度条和色度条）
Reset	初始化当前显示的映射函数并取消对所有曲线参数的选定
Reset All	初始化所有映射函数
Input/Output	显示变换曲线上选取的控制点的坐标，Input指水平轴，Output指垂直轴
Component	为交互式操作选择映射函数。在RGB空间中，选项包括R，G，B和RGB（映射所有三个彩色分量）。在HIS空间中，选项是H，S，I和HIS，依次类推



(a) (b)

图6.13 (a)负映射函数；(b)该函数作用于图6.12所示单色图像后的效果

逆或负映射函数在彩色处理中也很有用。正如在图6.14(a)和图6.14(b)中所看到的那样，映射的结果使人回想起了普通彩色胶片的负片。例如，图6.14(a)中最底行的粉笔的红色，被变换为图6.14(b)中的青色，即红色的补色。原色的补色是其他两种原色的混和（例如，青色就是蓝加绿）。在灰度情况下，补色可用于增加嵌入在暗色区域中的细节，尤其是当这些区域的面积较大时。注意，在图6.13(a)中的全彩色条包含图6.12中所示全彩条的色调的补色。



(a) (b)

图6.14 (a)全彩色图像；(b)它的负片（彩色补色）

例6.4 单色和彩色对比度增强

下面考虑进行单色和彩色对比度操作时函数ice的用途。图6.15(a)到图6.15(c)显示了处理单色图像时函数ice的效果。图6.15(d)到图6.15(f)显示了处理彩色图像时函数ice的效果。正如前一个例子所示，未显示的映射函数仍保持为默认值或1:1状态。在两个处理事件中，启用了Show PDF复选框。这样，图6.15(a)所示的航空照片的直方图就显示在图6.15(c)中的(形映射函数之下(参见3.2.1节)；图6.15(f)为图6.15(d)所示的彩色图像提供了三个直方图——对三个彩色分量中的每一个提供一个直方图。虽然图6.15(f)中的S形映射函数增加了图6.15(d)的对比度[与图6.15(e)相比]，但它对色度也有一些效果。色彩的微小改变在图6.15(e)中很难看到，但映射的结果是很明显的，正如我们在图6.15(f)所示的映射的Full-color Bar中看到的那样。回忆前一个例子可知，一幅RGB图像的三个分量的相同变化会对色彩产生戏剧性的效果(见图6.14中的彩色分量映射)。

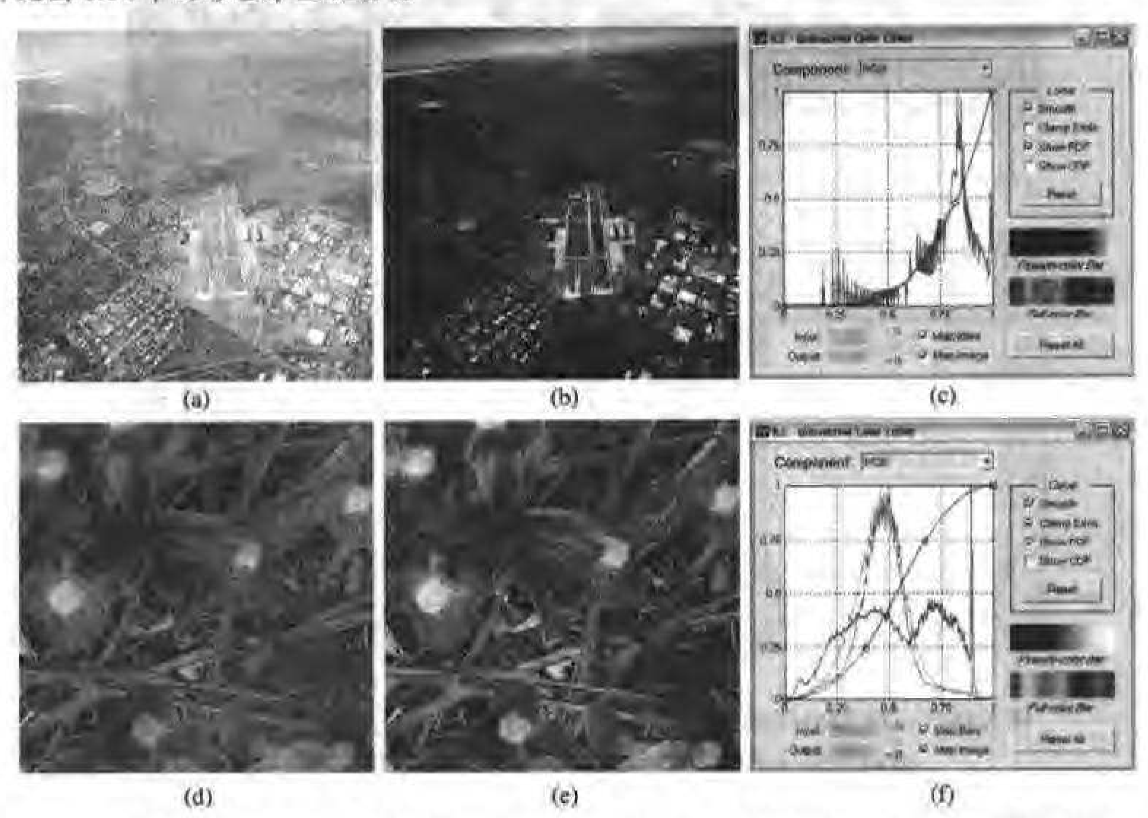


图6.15 使用函数ice增强单色和全彩色图像的对比度：(a)和(d)是输入图像，(b)和(e)显示了处理后的结果，(c)和(f)是ice显示(本例的黑白图像由NASA提供)

例6.3和例6.4中输入图像的红、绿和蓝分量施加了相同的映射，即使用了相同的变换函数。为避免说明三个相同的函数，函数ice提供了一个"全分量"函数(当在RGB彩色空间内操作时，指RGB曲线)，它用于映射所有的输入分量。下面的几个例子显示了其中三个分量进行不同处理的变换。

例6.5 伪彩色映射

正如早些时候提到的那样，当一幅黑白图像在RGB彩色空间中显示且单独对产生的分量进行映射时，变换的结果就是一幅伪彩色图像，其中输入图像的灰度级被任意颜色代替。做这种处理的变换是很有用的，因为人眼可以辨认出上百万种颜色，但相对来说只能辨认出不多的灰度。这样，伪彩色映射就常常用来细微地改变灰度以便人眼更易察觉，或者突出重要的灰度级区域。事实上，伪彩色的主要应用就是图像中灰度级的判读。

图6.16(a)是一幅包含几个裂缝和孔（通过图像中间的明亮白条纹）的焊接（水平黑色区域）的X光图像。伪彩色图像在图6.16(b)中显示。它是使用图6.16(c)和图6.16(d)中的映射函数对输入进行RGB的绿和蓝分量映射而产生的。注意，伪彩色映射产生了令人惊异的视觉差别。GUI伪彩色条为合成映射提供了方便的视觉指南。如图6.16(c)和图6.16(d)所示，交互地指定的映射函数会把从黑到白的灰度变换为蓝和红之间的色调，黄色则为白色保留。当然，黄色也对应于焊接的裂缝和孔，它们在此例中是很重要的特性。

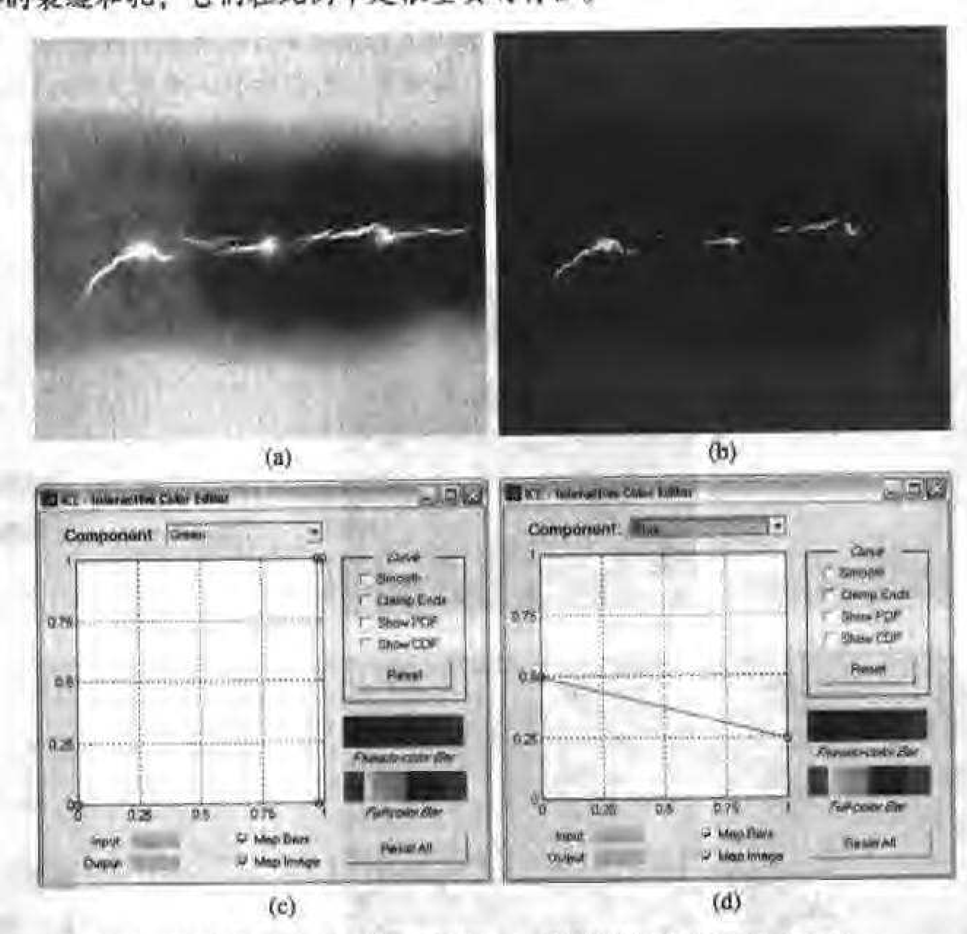


图6.16　(a)有缺陷焊接的X光图像；(b)焊接的伪彩色图像；(c)和(d)绿和蓝分量的映射函数（原像由X-TEK系统公司提供）

例6.6　彩色平衡

图6.17显示了涉及全彩图像的一个应用，它有助于独立地映射一幅图像的彩色分量。通常称为彩色平衡或彩色校正，这种类型的映射主要支持高端彩色再现系统，但现在可在大多数桌面电脑上运行。一种重要的应用是照片增强。虽然彩色不平衡可以通过颜色分光计对已知图像彩色的客观分析决定，但在RGB和CMY分量应该相等的白色区域存在时，正确的视觉估计是有可能的。在图6.17中可以看到，皮肤的特性对视觉估计来说是优秀的样本，因为人类对皮肤颜色很敏感。图6.17(a)显示了一幅母亲和孩子的CMY扫描图像，但其颜色偏品红（记住MATLAB只能显示图像的RGB版本）。为简单明了并与MATLAB兼容，函数ice也仅接受RGB（和单色）输入，但正如表6.4详述的那样，能在各种彩色空间中处理输入图像。为了交互地修改RGB图像f1的CMY分量，合适的ice调用是

```
>> f2 = ice('image', f1, 'space', 'CMY');
```

如图6.17所示，品红色的略微减少会对彩色图像造成重要影响。

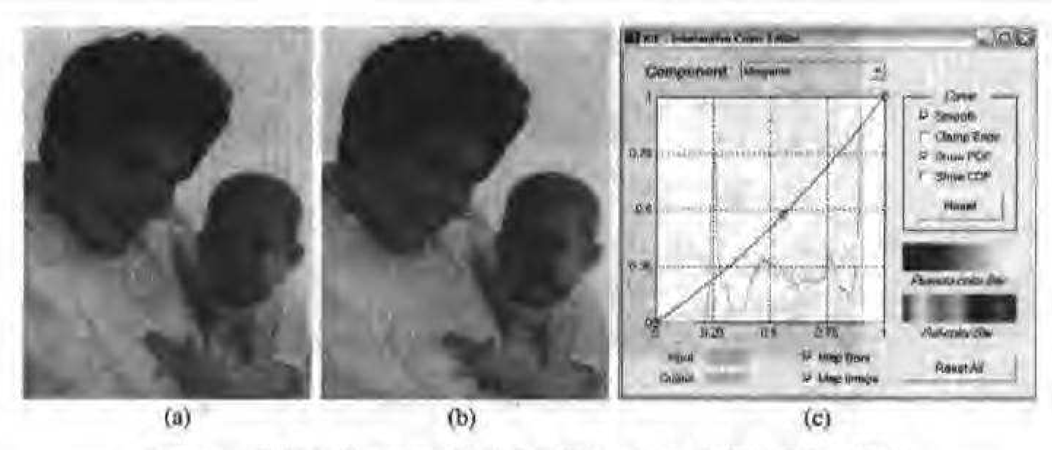


图 6.17　使用函数 ice 进行彩色平衡处理：(a)带有过重品红色的图像；(b)校正后的图像；(c)用于校正不平衡的映射函数

例 6.7　基于映射的直方图

直方图均衡化是灰度级映射处理，它寻求产生具有均匀亮度直方图的单色图像。正如在 3.3.2 节讨论过的那样，所需要的映射函数是输入图像中灰度级的累积分布函数（CDF）。因为彩色图像有很多分量，所以灰度技术必须进行修改，以便处理多个分量和相关的直方图。可以想得到，直方图分别均衡化彩色图像的分量是不明智的。其结果是经常产生错误的颜色。一种更合逻辑的方法是均匀地扩展彩色亮度，而保留彩色本身（如色调）不变。

图 6.18(a)显示了调味瓶架上放着调味瓶和搅拌机的彩色图像。图 6.18(b)显示的变换后图像明显要亮一些，该图像是使用图 6.18(c)和 6.18(d)中的 HSI 变换产生的。木桌的造型和纹理及放置在木桌上的几个调味瓶现在都看得见了。使用图6.18(c)中的函数映射了亮度分量，这相当于近似分量的 CDF（也显示在图上）。选择图 6.18(d)中的色调映射函数旨在改进亮度均衡化结果的彩色感观效果。注意，输入图像的直方图与输出图像的色调、饱和度和亮度分量分别显示在图 6.18(e)和图 6.18(f)中。当亮度和饱和度分量已经更改时，色调分量实际上还是完全相同的。注意，要在 HSI 彩色空间中处理一幅 RGB 图像，在 ice 的调用中，应输入合适的名称/数值对，即 'space'/'hsi'。

在这一节中，前几个例子中产生的输出图像是 RGB 图像和 uint8 类图像。对于例 6.3 所示的黑白结果，RGB 输出的三个分量是完全相同的。更简洁的表示可以通过表 6.3 中的函数 rgb2gray 或使用命令

```
f3 = f2(:, :, 1);
```

得到。其中，f2 是一幅由函数 ice 产生的 RGB 图像，f3 是一幅标准的 MATLAB 黑白图像。

6.5　彩色图像的空间滤波

6.4 节的材料在单个彩色分量平面的单个图像像素上执行彩色变换。更复杂级别的处理包括空间邻域处理，这也在单个图像平面上进行。这类似于 3.2 节中关于亮度变换的讨论和 3.4 节与 3.5 节中有关空间滤波的讨论。我们对彩色图像的空间滤波的介绍集中在 RGB 图像上，但对于其他彩色模型，这种基本概念也是可用的。下面我们使用两个线性滤波的例子来说明彩色图像的空间滤波：图像平滑和图像锐化。

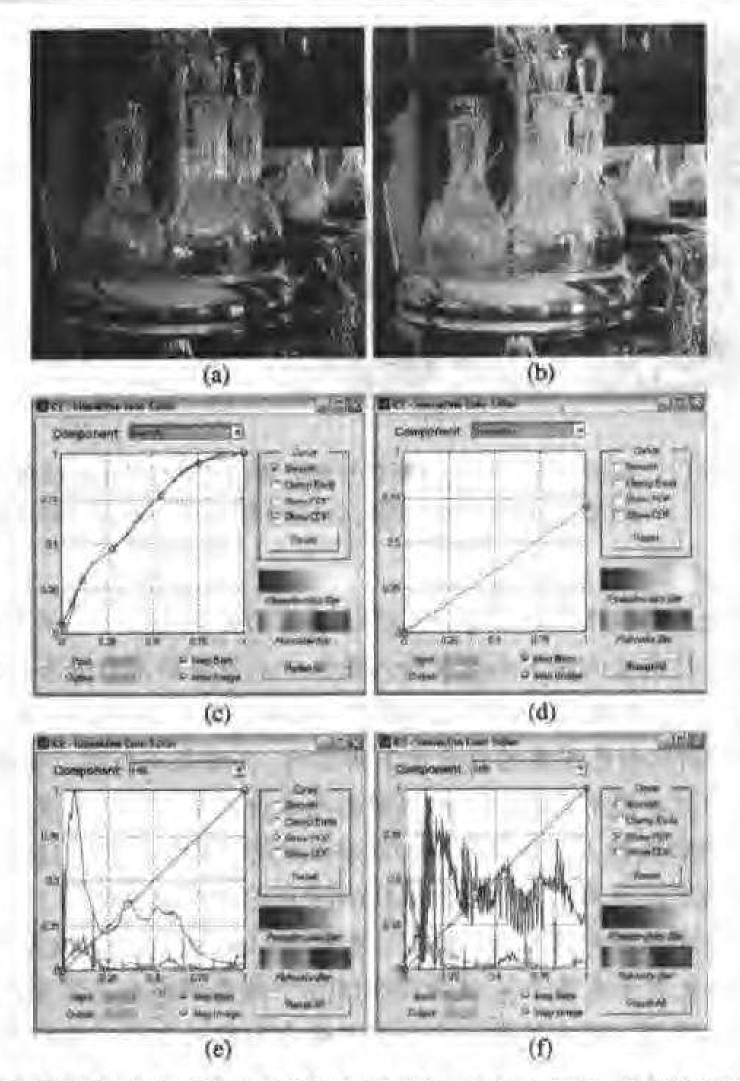


图6.18　在HSI彩色空间中进行直方图均衡化和饱和度调整：(a)输入图像；(b)映射结果；(c)亮度分量映射函数和累积分布函数；(d)饱和度分量映射函数；(e)输入图像的分量直方图；(f)映射结果的分量直方图

6.5.1　彩色图像平滑

参考图6.10(a)和3.4节及3.5节的讨论可知，单色图像的平滑（空间平均）可以通过空间掩模中的相应系数（全是1）去乘所有像素的值，并用掩模中元素的总数去除来实现。用空间掩模平滑全彩色图像的处理示于图6.10(b)中。该处理（如在RGB空间）用处理灰度图像的相同方法来表达，只是替代单个像素，我们现在处理示于6.3节的向量值。

令S_{xy}表示彩色图像中以(x, y)为中心的邻域的一组坐标。在该邻域中RGB向量的平均值是

$$\bar{\mathbf{c}}(x, y) = \frac{1}{K}\sum_{(s,t)\in S_{xy}} \mathbf{c}(s, t)$$

其中，K是邻域中像素的数量。它遵从6.3节中的讨论和向量加属性，即

$$\bar{\mathbf{c}}(x, y) = \begin{bmatrix} \dfrac{1}{K}\displaystyle\sum_{(s,t)\in S_{xy}} R(s, t) \\ \dfrac{1}{K}\displaystyle\sum_{(s,t)\in S_{xy}} G(s, t) \\ \dfrac{1}{K}\displaystyle\sum_{(s,t)\in S_{xy}} B(s, t) \end{bmatrix}$$

我们意识到该向量的每个分量都将作为我们希望得到的结果，该结果是对每一个分量图像执行邻域平均获得的，使用的是标准的灰度级邻域处理。从而我们可得出这样的结论：用邻域平均的平滑可以在独立分量的基础上进行。若邻域平均直接在彩色向量空间进行，则其结果是相同的。

如3.5.1节中讨论的那样，用于图像平滑的IPT线性空间滤波器是用函数 fspecial 产生的，该函数带有三个选项之一：'average'、'disk' 和 'gaussian'（见表3.4）。一旦产生了滤波器，就可使用函数 imfilter 执行滤波，详见3.4.1节中的讨论。

概念上，使用线性空间滤波器平滑RGB彩色图像 fc 的步骤如下。

1. 提取三幅分量图像：

```
>> fR = fc(:, :, 1); fG = fc(:, :, 2); fB = fc(:, :, 3);
```

2. 分别对每幅分量图像滤波。例如，令 w 表示使用 fspecial 产生的平滑滤波器，则平滑红色分量图像的方法如下：

```
>> fR_filtered = imfilter(fR, w);
```

其他两幅分量图像的平滑方法与此类似。

3. 重建滤波后的RGB图像：

```
>> fc_filtered = cat(3, fR_filtered, fG_filtered, fB_filtered);
```

然而，我们可以在MATLAB中执行RGB图像的线性滤波，所用的语法与单色图像所用的语法相同，这允许我们把前面的三个步骤合并为一步：

```
>> fc_filtered = imfilter(fc, w);
```

例6.8　彩色图像平滑

图6.19(a)显示了一幅有着1197 × 1197个像素的RGB图像，图6.19(b)到图6.19(d)是其RGB分量图像，它们是用前一段描述的步骤来提取的。图6.20(a)到图6.20(c)显示了图6.19(a)的三个HSI分量图像，它们是使用函数 rgb2hsi 得到的。

图6.21(a)显示了图6.19(a)所示图像平滑后的结果，平滑使用了带有 'replicate' 选项的 imfilter 函数和大小为25 × 25像素的 'average' 滤波器。平均滤波器较大，足以产生有意义的模糊度。选择该尺寸滤波器的目的在于，证明在RGB空间的平滑和仅用被变换到HSI空间后图像的亮度分量达到的结果之间的不同。图6.21(b)是使用如下命令获得的：

```
>> h = rgb2hsi(fc);
>> H = h(:, :, 1); S = h(:, :, 2); I =  h(:, :, 3);
>> w = fspecial('average', 25);
>> I_filtered = imfilter(I, w, 'replicate');
>> h = cat(3, H, S, I_filtered);
>> f = hsi2rgb(h);
>> f = min(f, 1); % RGB images must have values in the range [0, 1].
>> imshow(f)
```

显然，两种滤波后的结果是十分不同的。例如，除图像有点模糊以外，图6.21(b)中花朵的顶部还出现了绿色边缘。原因很简单，即色调和饱和度分量没有变化，而平滑处理使得亮度分量的值的变化得以减少。合乎逻辑的情况是用相同的滤波器去平滑所有三个分量。然而，这将改变色调和饱和度之间的相对关系，从而产生无意义的颜色，如图6.21(c)所示。

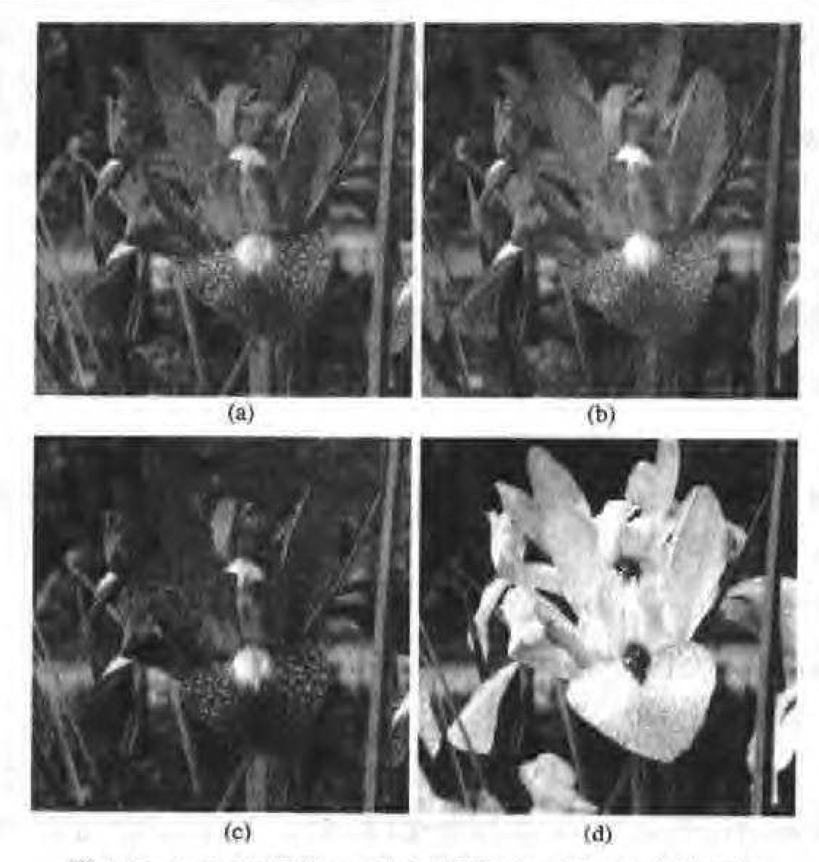


图 6.19 (a)RGB 图像，(b)到(d)分别是红、绿、蓝分量图像

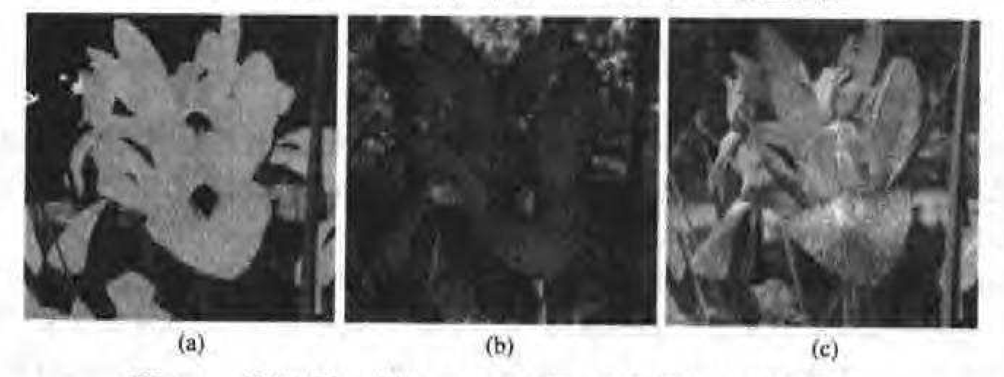


图 6.20 从左到右：图 6.19(a)的色调、饱和度和亮度分量图像

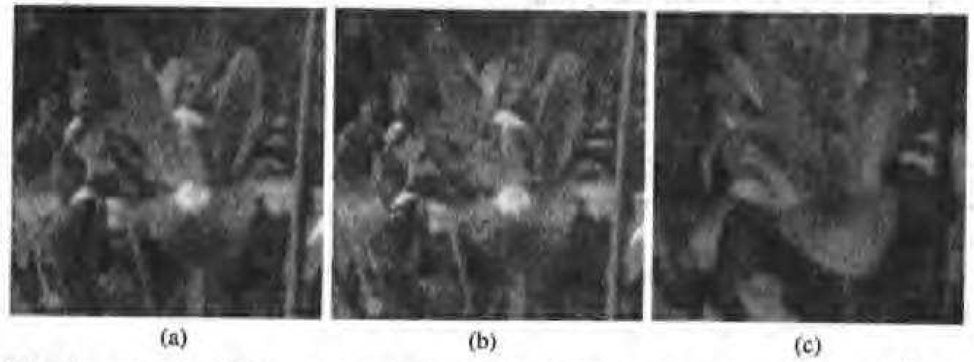


图 6.21 (a)分别平滑 R，G 和 B 图像平面得到的平滑后的 RGB 图像；(b)仅平滑 HSI 图像的亮度分量后的结果；(c)平滑所有三个 HSI 分量后的结果

一般来说，当掩模的尺寸减小时，对 RGB 分量图像进行滤波和对同一幅 HSI 图像的亮度分量进行滤波时，得到的差别也减少了。

6.5.2 彩色图像锐化

用线性空间滤波器锐化一幅RGB彩色图像遵循前节所示的相同步骤，但使用的应是锐化滤波器。在本节中，我们考虑使用拉普拉斯算子（见3.5.1节）来使图像锐化。从向量分析中，我们知道一个向量的拉普拉斯算子定义为一个向量，其分量等于输入向量的标量分量的拉普拉斯算子。在RGB彩色制式中，在6.3节中引入的向量c的拉普拉斯算子是

$$\nabla^2[\mathbf{c}(x,y)] = \begin{bmatrix} \nabla^2 R(x,y) \\ \nabla^2 G(x,y) \\ \nabla^2 B(x,y) \end{bmatrix}$$

如前一节介绍的那样，它告诉我们可以通过分别计算每个分量图像的拉普拉斯算子来计算全彩色图像的拉普拉斯算子。

例6.9 彩色图像锐化

图6.22(a)显示了图6.19(a)所示图像的模糊版本 fb，它是用一个5×5均值滤波器得到的。为锐化这幅图像，可使用拉普拉斯算子滤波器掩模

```
>> lapmask = [1 1 1; 1 -8 1; 1 1 1];
```

然后，如例3.9那样，使用如下命令计算并显示增强后的图像：

```
>> fen = imsubtract(fb, imfilter(fb, lapmask, 'replicte'));
>> imshow(fen)
```

这里，我们已将两个需要的步骤合并成为一个命令。如前一节那样，在使用 imfilter 时，RGB图像完全被当做单色图像来处理（如有着相同的语法）。图6.22(b)显示了结果。注意水滴、叶脉、花朵的黄色中心和前景中的绿色植物等的锐化增强效果。



(a) (b)

图6.22 (a)模糊图像；(b)用拉普拉斯算子增强图像，而后再用函数 ice 增强对比度

6.6 在RGB向量空间直接处理

如在6.3节中提到的那样，存在这样一些情况，即基于单独彩色平面的处理不等于直接在RGB向量空间中的处理。在本节中，我们将通过考虑彩色图像处理的两个重要应用来说明向量处理：彩色边缘检测和区域分割。

于二维空间，但是不能扩展到高维空间。将其运用到RGB图像的惟一方法是计算每个彩色分量图像的梯度，然后合并该结果。遗憾的是，如在该节稍后说明的那样，这与直接在RGB向量空间中计算边缘是不同的。

然后，问题是定义在6.3节中已定义过的向量**c**的梯度。下面是把梯度的概念扩展到向量函数的各种方法之一。标量函数$f(x, y)$的梯度是一个在坐标(x, y)处f的最大变化率的方向上的向量。

令**r**, **g**和**b**是RGB彩色空间沿R，G和B轴的单位向量（见图6.2），并定义向量

$$\mathbf{u} = \frac{\partial R}{\partial x}\mathbf{r} + \frac{\partial G}{\partial x}\mathbf{g} + \frac{\partial B}{\partial x}\mathbf{b}$$

和

$$\mathbf{v} = \frac{\partial R}{\partial y}\mathbf{r} + \frac{\partial G}{\partial y}\mathbf{g} + \frac{\partial B}{\partial y}\mathbf{b}$$

令g_{xx}，g_{yy}和g_{xy}是这些向量的点积，如下所示：

$$g_{xx} = \mathbf{u}\cdot\mathbf{u} = \mathbf{u}^T\mathbf{u} = \left|\frac{\partial R}{\partial x}\right|^2 + \left|\frac{\partial G}{\partial x}\right|^2 + \left|\frac{\partial B}{\partial x}\right|^2$$

$$g_{yy} = \mathbf{v}\cdot\mathbf{v} = \mathbf{v}^T\mathbf{v} = \left|\frac{\partial R}{\partial y}\right|^2 + \left|\frac{\partial G}{\partial y}\right|^2 + \left|\frac{\partial B}{\partial y}\right|^2$$

和

$$g_{xy} = \mathbf{u}\cdot\mathbf{v} = \mathbf{u}^T\mathbf{v} = \frac{\partial R}{\partial x}\frac{\partial R}{\partial y} + \frac{\partial G}{\partial x}\frac{\partial G}{\partial y} + \frac{\partial B}{\partial x}\frac{\partial B}{\partial y}$$

记住，R，G和B以及g项是x和y的函数。通过使用这种表示，可以看出（见Di Zenzo[1986]）作为(x, y)的函数$\mathbf{c}(x, y)$的最大变化率的方向由角度

$$\theta(x, y) = \frac{1}{2}\arctan\left[\frac{2g_{xy}}{(g_{xx} - g_{yy})}\right]$$

给出，并且变化率的值（例如梯度值）在由$\theta(x, y)$的元素给出的方向上由

$$F_\theta(x, y) = \left\{\frac{1}{2}\left[(g_{xx} + g_{yy}) + (g_{xx} - g_{yy})\cos 2\theta + 2g_{xy}\sin 2\theta\right]\right\}^{1/2}$$

给出。注意，$\theta(x, y)$和$F_\theta(x, y)$是与输入图像大小相同的图像。$\theta(x, y)$的元素是在计算梯度后每个点的角度，$F_\theta(x, y)$是梯度图像。

由于$\tan(\alpha) = \tan(\alpha \pm \pi)$，若$\theta_0$是前述arctan方程的一个解，则$\theta_0 \pm \pi/2$也是该方程的一个解。此外，由于$F_\theta(x, y) = F_{\theta+\pi}(x, y)$，所以$F$仅需在半开区间$[0, \pi)$上计算$\theta$的值。arctan方程提供两个相隔90°的值意味着该方程与两个正交方向上的每个点(x, y)相关。沿着这两个方向之一F最大，而沿着另一个方向F最小，所以最终结果是由选择每个点上的最大值产生的。这些结果的推导很长，对当前讨论而言进行这种推导意义不大。感兴趣的读者可以参考Di Zenzo[1986]。执行前述方程要求计算偏导数，如本节早先讨论过的Sobel算子。

下面的函数用于实现RGB图像的彩色梯度（代码见附录C）：

```
[VG, A, PPG] = colorgrad(f, T)
```

其中`f`是一幅RGB图像，`T`是范围[0, 1]内的一个可选阈值（默认为0）；`VG`是RGB向量梯度$F_\theta(x, y)$；`A`是以弧度计的角度，用于实现RGB图像的彩色梯度$\theta(x, y)$；`PPG`是对单独彩色平面（为比较目的而产生）的二维梯度求和形成的梯度。这些梯度符号是$\nabla R(x, y)$、$\nabla G(x, y)$和$\nabla B(x, y)$，运算符号∇，已在本节前面定义。实现前述方程所要求的全部导数可在函数`colorgrad`中用Sobel算子实现。输

出VG和PPG通过函数colorgrad归一化到范围[0, 1]内，并且它们被阈值处理，以便其值小于或等于T时VG(x,y) = 0，其他情况下VG(x,y) = VG(x,y)。PPG的情况与此类似。

例 6.10 用函数 colorgrad 进行 RGB 边缘检测

图6.24(a)到图6.24(c)显示了三幅简单的黑白图像，当使用RGB平面时，会产生图6.24(d)所示的彩色图像。这个例子的目的是(1)举例说明函数colorgrad的应用，(2)证明通过合并单独彩色平面的梯度来计算彩色图像的梯度与直接在RGB向量空间中用刚刚讨论的方法来计算梯度是不同的。

令f表示图6.24(d)中的RGB图像，命令

```
>> [VG, A, PPG] = colorgrad(f);
```

产生图6.24(e)和图6.24(f)所示的图像VG和PPG。这两个结果间最重要的不同是图6.24(f)中的水平边缘比图6.24(e)中的对应边缘更弱。其原因很简单：红色和绿色平面的梯度［见图6.24(a)和图6.24(h)］产生两个垂直边缘，蓝色平面的梯度产生单个水平边缘。为形成PPG而相加这三个梯度会产生亮度两倍于水平边缘亮度的垂直边缘。

另一方面，当彩色图像的梯度在向量空间中［见图6.24(e)］直接计算时，垂直和水平边缘的比值是$\sqrt{2}$而不是2。其原因也很简单：参考图6.2(a)所示的彩色立方体和图6.24(d)所示的图像，我们看到彩色图像的垂直边缘在彩色图像中位于蓝白方块和黑黄方块之间。这些颜色在彩色立方体之间的距离是$\sqrt{2}$，但是在黑蓝和黄白（水平边缘）之间的距离仅是1。从而垂直和水平比率是$\sqrt{2}$。若边缘精度存在问题，尤其是在使用阈值时，则这两种方法之间的差别会很大。例如，若使用一个大小为0.6的阈值，则图6.24(f)中的水平线将会消失。

在实际中，当我们的兴趣在边缘检测上而不是精度上时，两种刚刚讨论过的方法一般会产生差不多的结果。例如，图6.25(b)和图6.25(c)类似于图6.24(e)和图6.24(f)。它们通过对图6.25(a)应用函数colorgrad来获得。图6.25(d)是两个梯度图像的差，标度范围为[0, 1]。两幅图像的绝对最大差是0.2，相当于在8比特范围[0, 255]上将其平移到灰度级51。然而，这两个梯度图像在视觉表现上却十分接近，只是图6.25(b)在某些地方有点亮（与前一段解释的理由相似）。这样，对于这种类型的分析，每个独立分量梯度的计算方法比较简单，一般来说可以接受。在其他情况下，可能需要更准确的向量方法。

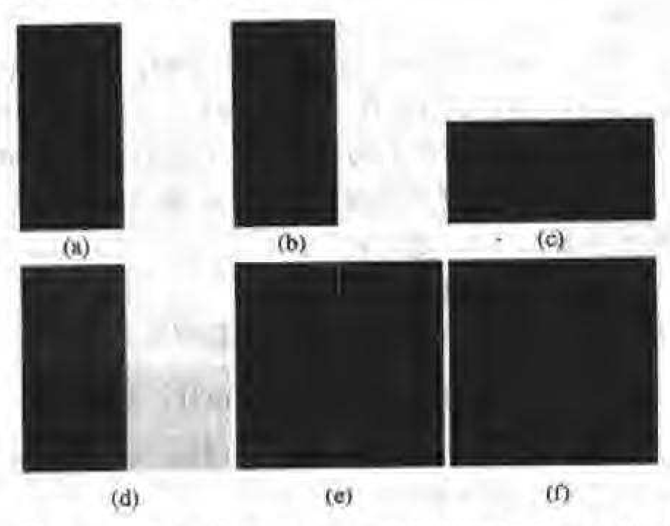


图6.24 (a)到(c)RGB分量图像（黑是0，白是255）；(d)相应的彩色图像；(e)在RGB向量空间中直接计算的梯度；(f)通过分别计算每个RGB分量图像的二维梯度并将结果相加而得到的合成梯度

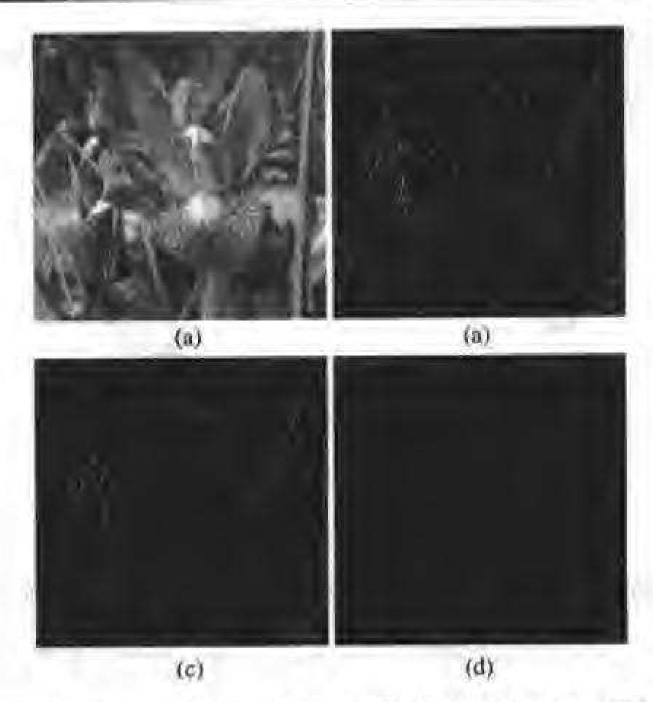


图6.25 (a)RGB图像；(b)在RGB向量空间中计算的梯度；(c)如图6.24(f)那样计算的梯度；(d)(b)和(c)的绝对差，标度范围为[0, 1]

6.6.2 RGB向量空间中的图像分割

分割是把一幅图像分成一些区域的处理。虽然分割是第10章的主题，但为连续起见，这里概要地考虑一下彩色区域分割。下面的讨论对读者来说不会有困难。

使用RGB彩色向量进行彩色区域分割是很简单的。假设我们的目的是在RGB图像中分割一个特定彩色范围的物体。给定一组感兴趣的彩色（或彩色范围）描述的彩色样本点，我们获得一个"平均"的颜色估计，它是我们希望分割的那种颜色。让这种平均色用RGB列向量$\mathbf{m}$来定义。分割的目的是对图像中的每一个RGB像素进行分类，使其在指定的范围内有一种颜色或没有颜色。为执行这一比较，我们需要一个相似性度量。最简单的度量之一是欧几里得距离。令$\mathbf{z}$表示RGB空间的任意点。若$\mathbf{z}$和$\mathbf{m}$之间的距离小于指定的阈值T，则我们说$\mathbf{z}$相似于$\mathbf{m}$。$\mathbf{z}$和$\mathbf{m}$之间的欧几里得距离由下式给出：

$$\begin{aligned}D(\mathbf{z},\mathbf{m}) &= \|\mathbf{z}-\mathbf{m}\| \\ &= \left[(\mathbf{z}-\mathbf{m})^T(\mathbf{z}-\mathbf{m})\right]^{1/2} \\ &= \left[(z_R-m_R)^2+(z_G-m_G)^2+(z_B-m_B)^2\right]^{1/2}\end{aligned}$$

其中$\|\cdot\|$是参量的范数，下标R、G和B表示向量$\mathbf{m}$和$\mathbf{z}$的RGB分量。$D(\mathbf{z},\mathbf{m})\leqslant T$的点的轨迹是一个半径为$T$的实心球体，如图6.26(a)所示。由定义可知，包含在球体内部或表面的点满足特定的彩色准则；而球体外面的点则不满足。在图像中对这两组点编码，如黑的和白的，产生一幅二值分割图像。

前述方程的一个有用归纳是距离

$$D(\mathbf{z},\mathbf{m}) = \left[(\mathbf{z}-\mathbf{m})^T\mathbf{C}^{-1}(\mathbf{z}-\mathbf{m})\right]^{1/2}$$

其中，$\mathbf{C}$是我们要分割的彩色的样值表示的协方差矩阵①。该距离称为Mahalanobis距离。$D(\mathbf{z},\mathbf{m})\leqslant T$的点的轨迹描述了一个实心三维椭圆体[见图6.26(b)]，它的重要属性是其主轴取在最大的数据扩

① 向量样本集的协方差矩阵的计算在11.5节讨论。

展方向上。当C等于单位矩阵I时，Mahalanobis距离约简为欧几里得距离。除了现在数据包含在椭球体内而不是包含在圆球体内之外，分割和在前段描述过的一样。

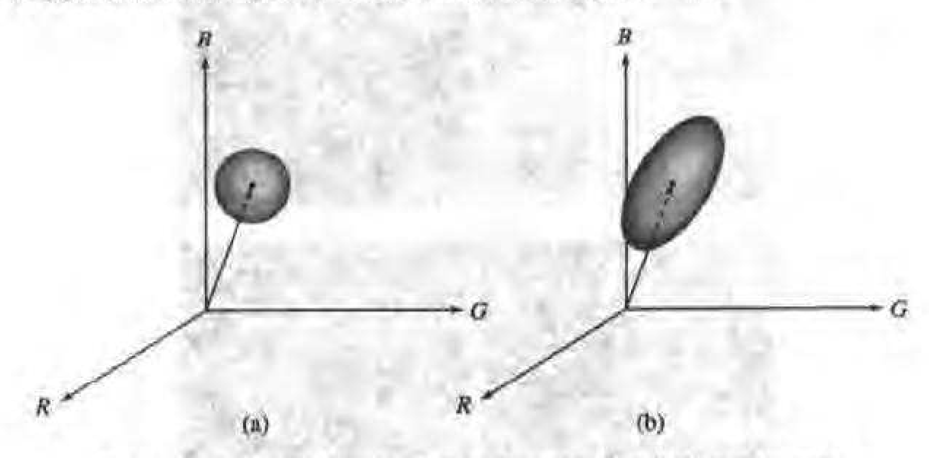


图6.26 为分割目的而在RGB向量空间中聚合数据的两种方法

在刚才描述的方法中，分割通过函数colorseg实现（代码见附录C），其语法为

```
S = colorseg(method, f, T, parameters)
```

其中，method 不是'euclidean'就是'mahalanobis'，f是待分割的RGB图像，T是前边描述过的阈值。若选择'euclidean'，则输入参数是m，若选择'mahalanobis'，则输入参数是m和C。参数m是一个在上面描述过的向量**m**，它的形式不是行就是列，并且C是3 × 3协方差矩阵**C**。输出S是一幅二值图像（和原始图像同样大小），在未通过阈值测试的点包含0，在通过了阈值测试的点包含1。1表示从基于彩色内容的f中分割的区域。

例6.11 RGB彩色图像分割

图6.27(a)显示的是木星卫星Io表面的一个区域的伪彩色图像。在这幅图像中，淡红色表示新从活火山喷射出来的熔岩，周围的黄色表示硫磺沉积。这个例子说明了使用函数colorseg的两个选项对淡红色区域的分割。

首先我们获得表示待分割彩色区域的样本。获得感兴趣区域的简单方法是使用5.2.4节中描述的函数roipoly，它产生一个交互选择的区域的二值掩模。从而，若让f表示图6.27(a)所示的彩色图像，则图6.27(b)中的区域可用如下命令得到：

```
>> mask = roipoly(f);          % Select region interactively.
>> red = immultiply(mask, f(:, :, 1));
>> green = immultiply(mask, f(:, :, 2));
>> blue = immultiply(mask, f(:, :, 3));
>> g = cat(3, red, green, blue);
>> figure, imshow(g)
```

其中，mask是一幅二值图像（大小与f相同），其背景为0，交互选择区域为1。

接着，我们计算ROI中的点的均值向量和协方差矩阵，但ROI中的点的坐标必须首先提取出来。

```
>> [M, N, K] = size(g);
>> I = reshape(g, M * N, 3); % reshape is discussed in Sec. 8.2.2.
>> idx = find(mask);
>> I = double(I(idx, 1:3));
>> [C, m] = covmatrix(I); % See Sec. 11.5 for details on covmatrix.
```

第二条语句将g中的彩色像素重新排列为I的行，第三条语句找出非黑彩色像素的行索引。这些都不是图6.27(b)中的掩模图像的背景像素。

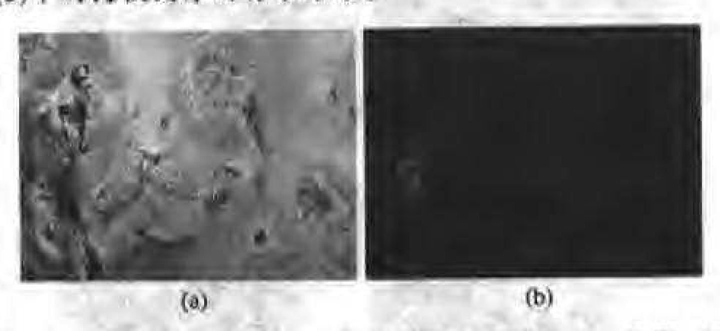


图6.27 (a)木星卫星Io表面的伪彩色图像；(b)使用函数roipoly交互地提取的感兴趣区域(原图由NASA提供)

最后的初步计算是决定T的值。首先让T变为一个彩色分量的标准偏差的倍数。C的主对角线包括RGB分量的方差，所以我们必须提取这些元素并计算它们的平方根：

```
>> d = diag(C);
>> sd = sqrt(d)'
   22.0643   24.2442   16.1806
```

sd的第一个元素是ROI中彩色像素的红色分量的标准偏差，另外两个分量与此类似。

现在进行图像分割，T值取25的倍数，这是最大标准偏差的近似：$T=25$，$T=50$，$T=75$，$T=100$，对于选项'euclidean'与$T=25$，我们使用

```
>> E25 = colorseg('euclidean', f, 25, m);
```

图6.28(a)显示了结果，并且图6.28(b)到图6.28(d)显示了用$T=50, 75, 100$分割的结果。类似地，图6.29(a)到图6.29(d)显示了使用相同的阈值顺序并使用'mahalanobis'选项获得的结果。有意义的结果[在图6.27(a)中取决于我们考虑为红色的内容]是用'euclidean'选项于$T=25$和50时得到的，但$T=75$和100时则产生了过度分割。另一方面，使用'mahalanobis'选项的结果对增加T值产生了更明显的变化。原因是在ROI中扩展的三维彩色数据在使用椭球体时会比使用圆球体时匹配得更好。注意，两种增加T的方法允许包含在分割区域中的红色更暗一些，而这正是我们所期望的。

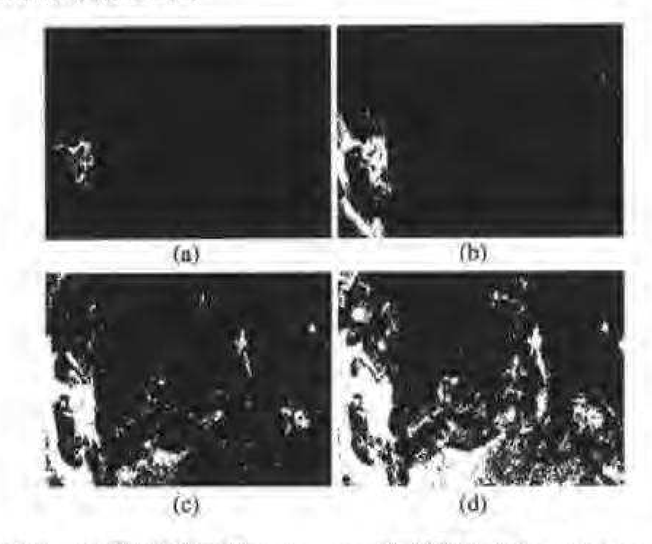


图6.28 (a)到(d)在函数colorseg中使用选项'euclidean'且T分别为25, 50, 75和100时，对图6.27(a)的分割

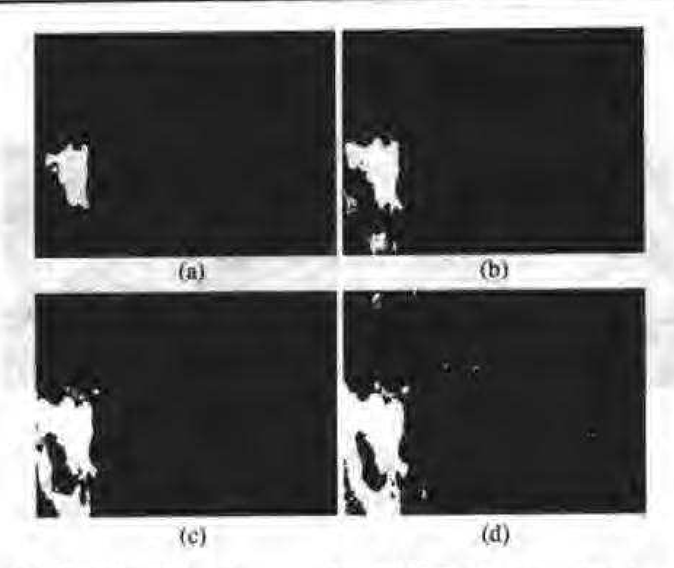


图6.29 (a)到(d)在函数colorseg中使用选项'mahalanobis'且T分别为25, 50, 75和100时，对图6.27(a)的分割。请与图6.28进行比较

小结

本章简要介绍了图像处理应用的基本主题以及图像处理中彩色的使用，并介绍了使用MATLAB、IPT以及前几节中开发的新函数来实现这些概念的方法。由于彩色模型的领域相当宽广，所以本书只关注这个主题。此处讨论的模型在图像处理中非常有用，因为它们为该领域的深入研究打下了良好的基础。

关于伪彩色的内容和关于单独彩色平面的全彩色处理，是前几章中为单色图像开发的图像处理技术的进一步拓展。彩色向量空间的探讨是从这些章中讨论的方法出发的，此外，我们还给出了灰度和全彩色图像处理间的一些重要差别。前几节中讨论的彩色向量处理技术是有代表性的基于向量的处理，包括中值和排序滤波、自适应和形态滤波、图像复原、图像压缩等。

第7章 小　　波

前言

当对数字图像进行多分辨率观察和处理时，离散小波变换（DWT）是首选的数学工具。除了具有有效、高度直观的描述框架以及多分辨率图像存储之外，DWT还有利于我们深入了解图像的空间域和频域特性，而傅里叶变换仅显示图像的频率特性。

在本章中，我们将探讨离散小波的计算和应用。我们将介绍小波工具箱，即一个为小波分析设计的MathWorks函数集，但不包括MATLAB的图像处理工具箱（IPT）和已开发的兼容子程序（允许单独使用IPT的基本小波处理）；换言之，没有小波工具箱。这些常用的函数和IPT相结合，提供了实现Gonzalez and Woods[2002]所著《数字图像处理》一书第7章讨论的所有概念。这些函数的用法大致相同，并且其功能与第4章中的IPT函数 `fft2` 和 `ifft2` 类似。

7.1　背景知识

考虑一个大小为 $M \times N$ 的图像 $f(x, y)$，其正向离散变换 $T(u, v, \ldots)$ 可用一般的多项式关系表示为

$$T(u, v, \ldots) = \sum_{x, y} f(x, y) g_{u, v, \ldots}(x, y)$$

其中，x 和 y 是空间变量，$u, v, \ldots$ 是变换域变量。若给定 $T(u, v, \ldots)$，则 $f(x, y)$ 可用一般的离散反变换

$$f(x, y) = \sum_{u, v, \ldots} T(u, v, \ldots) h_{u, v, \ldots}(x, y)$$

得到。$g_{u, v}$ 和 $h_{u, v}$…在这些方程中分别称为正变换核和反变换核。它们决定了变换对的性质、计算复杂度和主要用途。变换系数 $T(u, v, \ldots)$ 可看可做是 f 关于 $\{h_{u, v}, \ldots\}$ 的一系列展开系数。换言之，反变换核对于 f 的序列展开定义一组展开函数。

第4章的离散傅里叶变换（DFT）就与序列展开表示法完全吻合①。在这种情况下，

$$h_{u, v}(x, y) = g^*_{u, v}(x, y) = \frac{1}{\sqrt{MN}} e^{j2\pi(ux/M + vy/N)}$$

其中，$j = \sqrt{-1}$，* 是复共轭运算符，$u = 0, 1, \cdots, M-1$，且 $v = 0, 1, \cdots, N-1$。变换域的 v 和 u 分别表示水平和垂直频率。变换核是可分的，因为

$$h_{u, v}(x, y) = h_u(x) h_v(y)$$

其中，

$$h_u(x) = \frac{1}{\sqrt{M}} e^{j2\pi ux/M} \quad 和 \quad h_v(y) = \frac{1}{\sqrt{N}} e^{j2\pi vy/N}$$

---

① 第4章的DFT公式中，$1/MN$ 项放在反变换公式中。也可以将其只放在正变换中，或者如我们现在所做的那样，分开放在正反变换上，即 $1/\sqrt{MN}$。

是正交的，因为

$$\langle h_r, h_s\rangle = \delta_{rs} = \begin{cases}1 & r = s\\ 0 & 其他\end{cases}$$

其中〈 〉是内积运算符。变换核的可分性简化了二维变换的计算，因为这样我们就可以使用先行后列或先列后行的一维变换来实现二维变换；正交性导致了正反变换核之间的复共扼关系（若函数是实数，则它们相等）。

与离散傅里叶变换不同，离散傅里叶变换完全可以通过关于一对（前面已给出的）变换核的两个简单方程来定义，而术语“离散小波变换”指的是这样一类变换：不仅其中使用的变换核不同（例如，所用的展开函数），而且这些函数（例如，不管它们构成正交基还是双正交基）的基本特性和它们应用的方法（例如，计算多少不同的分辨率）也不同。因为DWT包含各种独特但相关的变换，所以我们不能写出一个能完全描述它们的公式。相反，我们利用变换核对或定义该核对的一组参数来表征每个 DWT。各种变换都与这样的事实有关，即变换的展开函数是变化频率和持续时间受限的“小波”（因此命名为小波）[见图7.1(b)]。在本章剩余的部分，我们将介绍几种小波核。每一个小波核都有如下的基本特性。

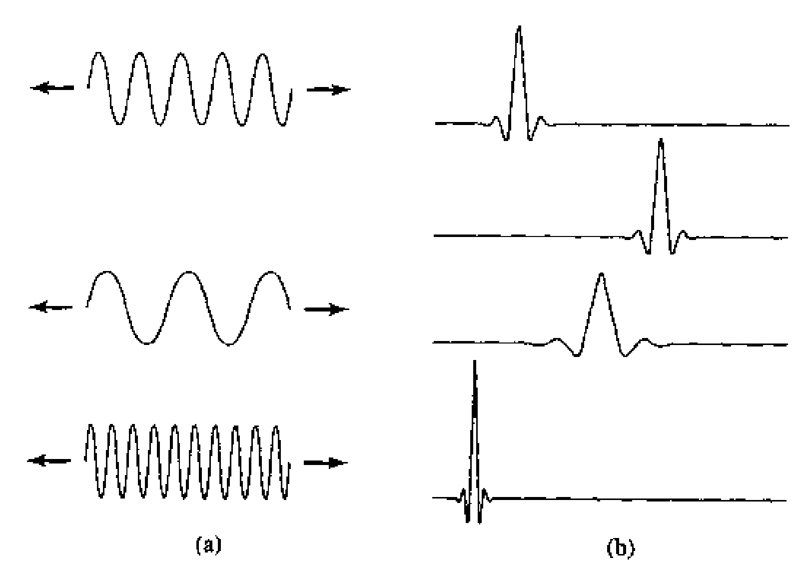


图 7.1 (a)傅里叶展开函数族是频率变化及持续时间无限的正弦波；(b)DWT 展开函数是持续时间有限及频率变化的“小波”

性质 1：可分离性、尺度可变性和平移性。核可用三个可分的二维小波来表示：

$$\psi^H(x, y) = \psi(x)\varphi(y)$$
$$\psi^V(x, y) = \varphi(x)\psi(y)$$
$$\psi^D(x, y) = \psi(x)\psi(y)$$

其中，$\psi^H(x, y)$, $\psi^V(x, y)$ 和 $\psi^D(x, y)$ 分别称为水平、垂直和对角小波，并且一个二维可分的尺度函数是

$$\varphi(x, y) = \varphi(x)\varphi(y)$$

每个二维函数是两个一维实平方可积的尺度和小波函数的乘积：

$$\varphi_{j,k}(x) = 2^{j/2}\varphi(2^j x - k)$$
$$\psi_{j,k}(x) = 2^{j/2}\psi(2^j x - k)$$

平移参数k决定了这些一维函数沿x轴的位置，尺度j决定了它们的宽度，即它们沿x轴有多宽多窄，而$2^{j/2}$控制它们的高度或振幅。注意，联合展开函数是母小波$\psi(x)=\psi_{0,0}(x)$和尺度函数$\varphi(x)=\varphi_{0,0}(x)$的二进制缩放和整数平移。

性质2：多分辨率的一致性。刚刚介绍的一维尺度函数满足多分辨率分析的如下需求：

a. $\varphi_{j,k}$与其整数平移正交。

b. 在低尺度或低分辨率（如较小的j）下可表示为一系列$\varphi_{j,k}$的展开的一组函数，包含在可以以更高尺度表示的那些函数中。

c. 惟一可以以任意尺度表示的函数是$f(x)=0$。

d. 当$j\to\infty$时，可用任何精度来表示任何函数。

当这些条件满足时，就存在一个伴随小波$\psi_{j,k}$及其整数平移和二进制尺度，其变化范围是在邻接尺度上可表示的任意两组函数$\varphi_{j,k}$之间的差。

性质3：正交性。展开函数（如$\{\varphi_{j,k}(x)\}$]对于一组一维可测的、平方可积函数形成一个正交基或双正交基。之所以称为基，是因为对于每一个可描述函数必须有惟一一组展开系数。正如在介绍傅里叶核时所说明的那样，对于实数来说正交核为$g_{u,v,\ldots}=h_{u,v,\ldots}$。对于双正交情况，

$$\langle h_r, g_s\rangle = \delta_{rs} = \begin{cases} 1 & r = s \\ 0 & \text{其他} \end{cases}$$

并且g称为h的对偶。对于利用尺度和小波函数$\varphi_{j,k}(x)$和$\psi_{j,k}(x)$的双正交小波变换，对偶分别定义为$\tilde{\varphi}_{j,k}(x)$和$\tilde{\psi}_{j,k}(x)$。

7.2 快速小波变换

上面性质的一个重要结果是$\varphi(x)$和$\psi(x)$可以用它们自身的双分辨率副本的线性组合来表达。这样，经过序列展开

$$\varphi(x) = \sum_n h_\varphi(n)\sqrt{2}\varphi(2x-n)$$

$$\psi(x) = \sum_n h_\psi(n)\sqrt{2}\varphi(2x-n)$$

其中，h_φ和h_ψ的展开系数分别称为尺度和小波向量。它们是快速小波变换（FWT）滤波器的系数，DWT的迭代计算方法示于图7.2。在图中，输出$W_\varphi(j,m,n)$和$\{W_\psi^i(j,m,n),\ i=H,V,D\}$是在尺度$j$处的DWT系数。方框中包含了时间反转尺度和小波向量，单元$h_\varphi(-n)$和$h_\psi(-m)$分别是低通和高通分解滤波器。最后，包括2和向下箭头的方框表示下取样，即从点的序列中每隔一个点来提取一个点。准确地说，用于计算图7.2中的$W_\psi^H(j,m,n)$的滤波和下取样操作是

$$W_\psi^H(j,m,n) = h_\psi(-m) * \left[h_\varphi(-n) * W_\varphi(j+1,m,n)|_{n=2k,\,k\geq 0}\right]|_{m=2k,\,k\geq 0}$$

其中，*表示卷积。在非负处计算卷积时，偶数索引等价于滤波和用2下取样。

每当输入通过图7.2中的滤波器组时，输入就会被分解为4个较低分辨率（或低尺度）的分量。W_φ系数是通过两个低通滤波器（如基于h_φ）产生的，因而称为近似系数；$\{W_\psi^i,\ i=H,V,D\}$分别是水平、垂直和对角线细节系数。因为$f(x,y)$是正变换图像的最高分辨率表示，所以第一次迭代的输入为$W_\varphi(j+1,m,n)$。注意，图7.2中的操作既不用小波也不用尺度函数，而只使用它们的相关

小波和尺度向量。另外，涉及了三个变换域变量，即尺度j、水平平移n和垂直平移m。这些变量与7.1节中的前两个公式中的$u, v, \ldots$相对应。

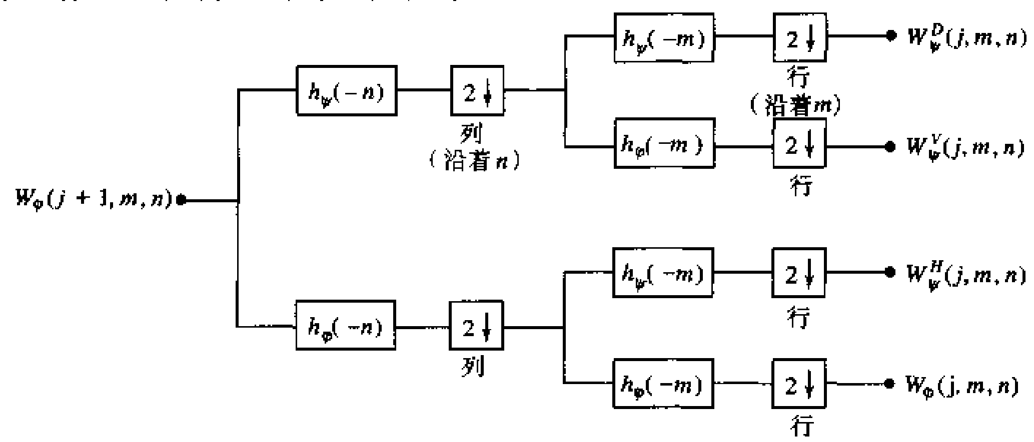


图7.2　二维快速小波变换（FWT）滤波器组。输入每通过一次将产生一个DWT尺度。在第一次迭代中，$W_\varphi(j+1, m, n) = f(x, y)$

7.2.1　使用小波工具箱的快速小波变换

在这一节中，我们将使用MATLAB小波工具箱来计算一幅大小为4×4的简单测试图像的FWT。在下一节中，我们将开发不使用小波工具箱来计算FWT的通用函数（如单独使用IPT）。此处仅提供一些开发这些函数的基本信息。

小波工具箱对于各种快速小波变换提供了分解滤波器。与特殊变换有关的滤波器可通过函数wfilters来访问，该函数的一般语法为

```
[Lo_D, Hi_D, Lo_R, Hi_R] = wfilters(wname)
```

其中，输入参数wname决定返回的滤波器系数，这些系数与表7.1中列出的一致；输出Lo_D、Hi_D、Lo_R和Hi_R是行向量，它们分别返回低通分解、高通分解、低通重构和高通重构滤波器（重构滤波器将在7.4节中讨论）。频繁地耦合的滤波器对可使用如下语句交替地检索：

```
[F1, F2] = wfilters(wname, type)
```

将type 设置为'd'、'r'、'l'或'h'将分别获得一对分解、重构、低通和高通滤波器。若执行这一条语句，则分解或低通滤波器在F1中返回，而重构或高通滤波器则放在F2中。

表7.1中列出了包含在小波工具箱中的FWT滤波器。它们的性质以及其他与尺度和小波函数相关的信息可在数字滤波和多分辨率分析方面的文献中找到。一些更重要的特性由小波工具箱的函数waveinfo和wavefun提供。例如，为了在MATLAB的命令窗口打印小波族wfamily（见表7.1）的描述，可在MATLAB提示符下输入

```
waveinfo(wfamily)
```

为了获得一个标准正交变换的尺度和/或小波函数的数字近似值，可键入

```
[phi, psi, xval] = wavefun(wname, iter)
```

它返回近似向量phi和psi，并且计算向量xval。正整数iter通过控制计算的迭代次数，可决定近似值的精度。对于双正交变换，合适的语法为

```
[phi1, psi1, phi2, psi2, xval] = wavefun(wname, iter)
```

其中，phi1和psi1是分解函数，而phi2和psi2是重构函数。

```
    Filters length              2
    Regularity                  haar is not continuous
    Symmetry                    yes
    Number of vanishing
    moments for psi             1

    Reference: I. Daubechies,
    Ten lectures on wavelets,
    CBMS, SIAM, 61, 1994, 194-202.
>> [phi, psi, xval] = wavefun('haar', 10);
>> xaxis = zeros(size(xval));
>> subplot(121); plot(xval, phi, 'k', xval, xaxis, '--k');
>> axis([0 1 -1.5 1.5]); axis square;
>> title('Haar Scaling Function');
>> subplot(122); plot(xval, psi, 'k', xval, xaxis, '--k');
>> axis([0 1 -1.5 1.5]); axis square;
>> title('Haar Wavelet Function');
```

图7.3显示了由最后6个命令产生的图形。函数title, axis和plot已在第2章和第3章中描述过，函数subplot用于把图形窗口细分为坐标轴的数组或子图，其语法为

```
H = subplot(m, n, p) or H = subplot(mnp)
```

其中，m和n分别是子图数组中的行数和列数。m和n都必须大于1。可选的输出变量H是由p选择的子图（例如轴）的句柄，根据p（从1开始）的增量值沿着图形窗口的顶行选择坐标轴，然后是第二行，依次类推。无论是否选择H，第p个轴都可以产生当前的图形。这样，前面给出的命令中的函数subplot(122)将选择1×2子图数组中的行1和列2的图形作为当前图形。然后，函数axis和title将作用于该图形。

示于图7.3中的Haar尺度和小波函数是不连续的和紧支撑的，这意味着在名为支撑的有限区间之外它们是零。注意，该支撑是1。另外，waveinfo数据说明了Haar展开函数是正交的，因此，正反变换核是相同的。

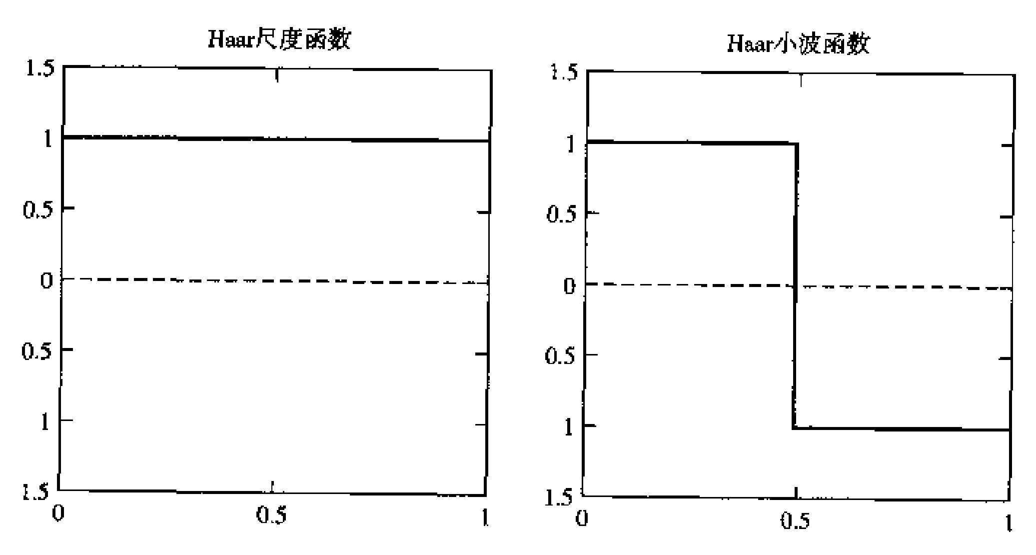


图7.3 Haar尺度函数和小波函数

给出一组分解滤波器，无论是用户提供的还是通过wfilters函数产生的，计算小波变换的最简方法是通过小波工具箱的函数wavedec2。该函数按如下方式调用：

```
[C, S] = wavedec2(X, N, Lo_D, Hi_D)
```

其中，X 是一幅二维图像或矩阵，N 是将被计算的尺度数（例如，通过图 7.2 所示 FWT 滤波器组的尺度数），Lo_D 和 Hi_D 是分解滤波器。更有效的语法为

```
[C, S] = wavedec2(X, N, wname)
```

其中，wname 也可以使用一个来自表 7.1 的假设值。输出数据结构[C,S]由行向量 C 组成（double 类），它包含计算出的小波变换系数以及定义了 C 中系数的排列的记录矩阵 S（也为 double 类）。C 和 S 之间的关系将在下一个例子中介绍，详细描述请参阅 7.3 节。

例 7.2　使用 Haar 滤波器的一个简单 FWT

考虑下面使用 Haar 小波的单尺度小波变换：

```
>> f = magic(4)
f =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

>> [c1, s1] = wavedec2(f, 1, 'haar')
c1 =
    Columns 1 through 9
        17.0000     17.0000     17.0000     17.0000     1.0000
        -1.0000     -1.0000      1.0000      4.0000

    Columns 10 through 16
        -4.0000     -4.0000      4.0000     10.0000     6.0000
        -6.0000    -10.0000

s1 =
     2     2
     2     2
     4     4
```

其中，一个大小为 4×4 的方形图像 f 变换为了一个大小为 1×16 的小波分解向量 c1 和一个大小为 3×2 的记录矩阵 s1。整个变换是通过单独执行（用 f 作为输入）图 7.2 中描述的操作而实现的。4 个 2×2 输出，产生一个下取样近似和三个方向（水平、垂直和对角线）的细节矩阵。函数 wavedec2 在行向量 c1 中以近似系数开始按列级联这些大小为 2×2 的矩阵，接着按水平、垂直和对角线级联这些矩阵。即 c1(1) 到 c1(4) 是近似系数 $W_\varphi(1,0,0)$，$W_\varphi(1,1,0)$，$W_\varphi(1,0,1)$ 和 $W_\varphi(1,1,1)$，图 7.2 中任意假定 f 的尺度为 2；c1(5) 到 c1(8) 是 $W_\psi^H(1,0,0)$，$W_\psi^H(1,1,0)$，$W_\psi^H(1,0,1)$，$W_\psi^H(1,1,1)$，等等。若从向量 c1 中提取水平细节系数矩阵，则可得到

$$W_\psi^H = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

记录矩阵 s1 提供矩阵的大小，该矩阵已被一次一列级联成了行向量 c1——加上原图像 f 的大小[在向量 s1(end,:) 中]。向量 s1(1,:) 和 s1(2,:) 分别包含计算出的近似矩阵的大小和三个细节系数矩阵。每个向量的第一个元素是参考细节或近似矩阵的行数，第二个元素是列数。当前面描述的单尺度变换扩展到两个尺度时，我们得到

```
>> [c2, s2] = wavedec2(f, 2, 'haar')
```

```
c2 =
    Columns 1 through 9
        34.0000        0            0          0.0000        1.0000
        -1.0000     -1.0000      1.0000       4.0000

    Columns 10 through 16
        -4.0000     -4.0000      4.0000      10.0000        6.0000
        -6.0000    -10.0000
s2 =
    1       1
    1       1
    2       2
    4       4
```

注意，c2(5:16) = c1(5:16)。元素c1(1:4)是单尺度变换的近似系数，这些系数已被填入图7.2的滤波器族，以便产生4个大小为1×1的输出：$W_{\varphi}(0,0,0)$, $W_{\psi}^{H}(0,0,0)$, $W_{\psi}^{V}(0,0,0)$, $W_{\psi}^{D}(0,0,0)$。这些输出按照在前面的单尺度变换中采用的相同的顺序按列级联起来（尽管此处它们是大小为1×1的矩阵），并代之以得到的近似系数。然后，列新记录矩阵s2，以反映这样一个事实，即c1中单一的2×2近似矩阵已由c2中的4个1×1的细节和近似矩阵所代替。这样，s2(end,:)又一次作为原图像的尺寸，s2(3,:)在尺度1的情况下是三个细节系数矩阵的大小，s2(2,:)在尺度0下是三个细节系数矩阵的大小，而s2(1,:)是最后近似的大小。

在这一节中，我们注意到，因为FWT是以数字滤波技术和卷积为基础的，所以边界失真会有所上升。为了将这些失真减到最小，该边界必须进行不同于图像其他部分的处理。当滤波器元素在卷积过程中落到图像外部时，必须为其值设定一个范围，该范围是图像外的滤波器的近似大小。包括函数wavedec2在内的许多小波工具箱函数根据全局参数dwtmode来扩展及填充待处理的图像。想要检查该动态扩展模式，可以输入st = dwtmode('status')，或者在MATLAB命令提示符下简单地输入dwtmode（如>>dwtmode）。要将STATUS设置为扩展模式，可以输入dwtmode(STATUS)；要使STATUS处于默认展开模式，可以使用语句dwtmode('save',STATUS)。表7.2中列出了支持的扩展模式和相应的STATUS值。

表7.2 小波工具箱图像扩展或填充模式

状态	描述
'sym'	图像由绕其边界的镜面反射来扩展，这一般是默认模式
'zpd'	图像由填充零值来扩展
'spd','sp1'	图像由一阶导数外推法扩展，或由最外面的两个边界值的线性填充来扩展
'sp0'	图像由外推边界值扩展，即通过复制边界值来扩展
'ppd'	图像由周期填充法扩展
'per'	图像在使用'sp0'扩展填充到偶数大小后（若可能），由周期填充法扩展

7.2.2 不使用小波工具箱的快速小波变换

在这一节中，我们将开发一对常用的函数，即函数wavefilter和wavefast，以代替前一节中提到的小波工具箱函数wfilters和wavedec2。我们的目的是深入讲解计算快速小波变换（FWT）的机制，同时开始为基于小波的图像处理建立不采用小波工具箱的“独立程序包”。这个过程将在7.3节和7.4节中给出，且由此得到的函数集可用来产生7.5节中的例子。

第一步是设计一个函数来产生小波分解和重构滤波器。下面这个称为wavefilter的函数利用标准的switch结构，与语句case和otherwise结合使用，可以按一种易于扩展的方式来完

成此项工作。尽管函数wavefilter仅提供了在《数字图像处理》(Gonzalez and Woods[2002])一书的第7章、第8章中验证过的滤波器，但通过附加适当的分解和重构滤波器（作为新情况），也可以适用于其他的小波变换。

```
function [varargout] = wavefilter(wname, type)
%WAVEFILTER Create wavelet decomposition and reconstruction filters.
%   [VARARGOUT] = WAVEFILTER(WNAME, TYPE) returns the decomposition
%   and/or reconstruction filters used in the computation of the
%   forward and inverse FWT (fast wavelet transform).
%
%   EXAMPLES:
%     [ld, hd, lr, hr] = wavefilter('haar') Get the low and highpass
%                                           decomposition(ld, hd)
%                                           and reconstruction
%                                           (lr, hr) filters for
%                                           wavelet 'haar'.
%     [ld, hd] = wavefilter('haar','d')     Get decomposition filters
%                                           ld and hd.
%     [lr, hr] = wavefilter('haar','r')     Get reconstruction
%                                           filters lr and hr.
%
%   INPUTS:
%     WNAME                         Wavelet Name
%     ---------------------------------------------------------------------%
%     'haar' or 'db1'               Haar
%     'db4'                         4th order Daubechies
%     'sym4'                        4th order Symlets
%     'bior6.8'                     Cohen-Daubechies-Feauveau biorthogonal
%     'jpeg9.7'                     Antonini-Barlaud-Mathieu-Daubechies
%
%     TYPE                          Filter Type
%     ---------------------------------------------------------------------%
%     'd'                           Decomposition filters
%     'r'                           Reconstruction filters
%
%   See also WAVEFAST and WAVEBACK.

% Check the input and output arguments.
error(nargchk(1, 2, nargin));

if (nargin == 1 & nargout ~= 4) | (nargin == 2 & nargout ~= 2)
   error('Invalid number of output arguments.');
end

if nargin == 1 & ~ischar(wname)
   error('WNAME must be a string.');
end

if nargin == 2 & ~ischar(type)
   error('TYPE must be a string.');
end

% Create filters for the requested wavelet.
switch lower(wname)
case {'haar', 'db1'}
   ld = [1 1]/sqrt(2);     hd = [-1 1]/sqrt(2);
   lr = ld;                hr = -hd;
```

```
end
% Output the requested filters.
if (nargin == 1)
   varargout(1:4) = {ld, hd, lr, hr};
else
   switch lower(type(1))
   case 'd'
      varargout = {ld, hd};
   case 'r'
      varargout = {lr, hr};
   otherwise
      error('Unrecognizable filter TYPE.');
   end
end
```

注意，对于函数wavefilter中的每个标准正交滤波器（即'haar'，'bd4'和'sym4'），重构滤波器是分解滤波器的时间反转形式，而高通分解滤波器则是其低通版本经调制后的形式。只有低通分解滤波器的系数需要在代码中明确给出，其他剩余的滤波器系数可以从中计算出来。在函数wavefilter中，时间反转是通过从最后一个到第一个来重新排列滤波器向量元素来实现的，语句为lr(end:-1:1) = ld。我们通过一个已知滤波器的分量与cos(pi*t)的乘积来完成调制，cos(pi*t)的值在-1到1之间变化，同时t从0开始增长而。对于函数wavefilter中的每个双正交滤波器（即'bior6.8'和'jepg9.7'），已指定了高通和低通分解滤波器；而重构滤波器可以由它们的调制计算出来。最后，我们注意到，由函数wavefilter生成的滤波器都有奇数长度。而且，零填充可用于确保每个小波的分解和重构滤波器的长度是相同的。

若给出函数wavefilter生成的一对分解滤波器，则可很容易地写出计算相关快速小波变换的通用子程序。我们的目标是设计一个基于滤波和如图7.2中的下取样操作的有效算法。为使该算法保留与现有小波工具箱的兼容性，我们采用同样的分解结构（即[C, S]，其中C表示一个分解向量，S表示一个记录矩阵）。下面这个名为wavefast的子程序，利用了对称图像扩展来减少与计算FWT相关联的边界失真：

```
function [c, s] = wavefast(x, n, varargin)
%WAVEFAST Perform multi-level 2-dimensional fast wavelet transform.
%   [C, L] = WAVEFAST(X, N, LP, HP) performs a 2D N-level FWT of
%   image (or matrix) X with respect to decomposition filters LP and
%   HP.
%
%   [C, L] = WAVEFAST(X, N, WNAME) performs the same operation but
%   fetches filters LP and HP for wavelet WNAME using WAVEFILTER.
%
%   Scale parameter N must be less than or equal to log2 of the
%   maximum image dimension. Filters LP and HP must be even. To
%   reduce border distortion, X is symmetrically extended. That is,
%   if X = [c1 c2 c3 ... cn] (in 1D), then its symmetric extension
%   would be [... c3 c2 c1 c1 c2 c3 ... cn cn cn-1 cn-2 ...].
%
%   OUTPUTS:
%     Matrix C is a coefficient decomposition vector:
%
%      C = [ a(n) h(n) v(n) d(n) h(n-1)  ... v(1) d(1) ]
%
```

```
%     where a, h, v, and d are columnwise vectors containing
%     approximation, horizontal, vertical, and diagonal coefficient
%     matrices, respectively. C has 3n + 1 sections where n is the
%     number of wavelet decompositions.
%
%     Matrix S is an (n+2) x 2 bookkeeping matrix:
%
%      S = [ sa(n,:) ; sd(n,:) ; sd(n-1,:) ; ... sd(1,:) ; sx ]
%
%     where sa and sd are approximation and detail size entries.
%
%   See also WAVEBACK and WAVEFILTER.

%   Check the input arguments for reasonableness.
error(nargchk(3, 4, nargin));

if nargin == 3
   if ischar(varargin{1})
      [lp, hp] = wavefilter(varargin{1}, 'd');
   else
      error('Missing wavelet name.');
   end
else
      lp = varargin{1}; hp = varargin{2};
end

fl = length(lp);      sx = size(x);

if (ndims(x) ~= 2) | (min(sx) < 2) | ~isreal(x) | ~isnumeric(x)
   error('X must be a real, numeric matrix.');
end

if (ndims(lp) ~= 2) | ~isreal(lp) | ~isnumeric(lp) ...
        | (ndims(hp) ~= 2) | ~isreal(hp) | ~isnumeric(hp) ...
        | (fl ~= length(hp)) | rom(fl, 2) ~= 0
   error(['LP and HP must be even and equal length real, ' ...
          'numeric filter vectors.']);
end

if ~isreal(n) | ~isnumeric(n) | (n < 1) | (n > log2(max(sx)))
   error(['N must be a real scalar between 1 and ' ...
          'log2(max(size((X))).']);
end

% Init the starting output data structures and initial approximation.
c = [];      s = sx;      app = double(x);

% For each decomposition ...
for i = 1:n
   % Extend the approximation symmetrically.
   [app, keep] = symextend(app, fl);

   % Convolve rows with HP and downsample. Then convolve columns
   % with HP and LP to get the diagonal and vertical coefficients.
   rows = symconv(app, hp, 'row', fl, keep);
   coefs = symconv(rows, hp, 'col', fl, keep);
   c = [coefs(:)' c]; s = [size(coefs); s];
   coefs = symconv(rows, lp, 'col', fl, keep);
```

```
    c = [coefs(:)' c];

    % Convolve rows with LP and downsample. Then convolve columns
    % with HP and LP to get the horizontal and next approximation
    % coefficients.
    rows = symconv(app, lp, 'row', fl, keep);
    coefs = symconv(rows, hp, 'col', fl, keep);
    c = [coefs(:)' c];
    app = symconv(rows, lp, 'col', fl, keep);
end

% Append final approximation structures.
c = [app(:)' c];      s = [size(app); s];

%-------------------------------------------------------------------%
function [y, keep] = symextend(x, fl)
% Compute the number of coefficients to keep after convolution
% and downsampling. Then extend x in both dimensions.

keep = floor((fl + size(x) - 1) / 2);
y = padarray(x, [(fl - 1) (fl - 1)], 'symmetric', 'both');

%-------------------------------------------------------------------%
function y = symconv(x, h, type, fl, keep)
% Convolve the rows or columns of x with h, downsample,
% and extract the center section since symmetrically extended.

if strcmp(type,'row')
   y = conv2(x, h);
   y = y(:, 1:2:end);
   y = y(:, fl / 2 + 1:fl / 2 + keep(2));
else
   y = conv2(x, h');
   y = y(1:2:end, :);
   y = y(fl / 2 + 1:fl / 2 + keep(1), :);
end
```

正如在主程序中看到的那样，仅有一个 for 循环，该循环根据产生的分解等级（或尺度），来组织完整的正变换计算。每执行一次循环，最初设置为 x 的当前近似图像 app 被内部函数 symextend对称地扩展。这个名为padarray的函数曾在3.4.2节中介绍过，它的功能是通过其穿过边界的元素的镜面反射 fl-1（即分解滤波器的长度减 1），将 app 扩展为二维。

函数symextend返回一个扩展过的近似系数矩阵，以及从随后的卷积和下取样结果的中心提取出来的像素数。扩展过的近似矩阵的行是用高通分解滤波器hp和经由函数symconv下取样的下一个卷积值。下一段中我们将描述这个函数。接着将卷积输出rows提交给函数symconv，利用hp和lp对它的列进行卷积和下取样操作，产生图7.2中最上边两个分支的对角线和垂直细节系数。这些结果将插入到分解向量c中（处理顺序为从最后一个元素到第一个元素）；同时，该程序重复与图 7.2 一致的过程来生成水平细节值和近似系数（图 7.2 最底部的两个分支）。

函数symconv运用函数conv2来完成大量的变换计算工作。该函数使滤波器h与x的行或列（取决于type）进行卷积，抛弃偶数索引行或列（即以2下取样），并提取每行或列的中心keep元素。使用矩阵 x 和行滤波器向量 h 来调用函数 conv2 可开始逐行卷积操作；面使用列滤波器向量 h' 会导致列卷积操作。

例7.3 比较函数wavefast 和函数wavedec2的执行时间

下面的测试程序利用函数tic和toc来比较小波工具箱函数wavedec2和通用函数wavefast的执行时间。

```
function [ratio, maxdiff] = fwtcompare(f, n, wname)
%FWTCOMPARE Compare wavedec2 and wavefast.
%   [RATIO, MAXDIFF] = FWTCOMPARE(F, N, WNAME) compares the operation
%   of toolbox function WAVEDEC2 and custom function WAVEFAST.
%
%   INPUTS:
%     F           Image to be transformed.
%     N           Number of scales to compute.
%     WNAME       Wavelet to use.
%
%    OUTPUTS:
%      RATIO      Execution time ratio (custom/toolbox)
%      MAXDIFF    Maximum coefficient difference.

% Get transform and computation time for wavedec2.
tic;
[c1, s1] = wavedec2(f, n, wname);
reftime = toc;

% Get transform and computation time for wavefast.
tic;
[c2, s2] = wavefast(f, n, wname);
t2 = toc;

% Compare the results.
ratio = t2 / (reftime + eps);
maxdiff = abs(max(c1 - c2));
```

对于图7.4所示的大小为512 × 512的图像和关于第四阶Daubechies小波的5尺度小波变换，函数fwtcompare得到

```
>> f = imread('vase', 'tif');
>> [ratio, maxdifference] = fwtcompare(f, 5, 'db4')

ratio =
    0.5508

maxdifference =
    3.2969e-012
```

注意，当产生实际上相同的结果时，自定义函数wavefast的速度要比对应的小波工具箱函数几乎快两倍。



图7.4 大小为512 × 512的花瓶图像

7.3 小波分解结构的运算

前两节中介绍的小波变换函数产生{c, S}形式的不可显示数据结构，其中c是变换系数向量，S是定义c中系数排列的记录矩阵。为了处理图像，我们必须能够检查和/或修改c。在这一节中，我们形式化地定义{c, S}，为了对其进行操作而检查一些小波工具箱函数，并且开发一些可以不借助小波工具箱而单独运用的自定义函数。这些函数可以用于创建显示c的通用程序。

例7.2中引入的表示方案可将多尺度二维小波变换系数集成为一个单一的一维向量

$$\mathbf{c} = [\mathbf{A}_N(:)' \quad \mathbf{H}_N(:)' \quad \cdots \quad \mathbf{H}_i(:)' \quad \mathbf{V}_i(:)' \quad \mathbf{D}_i(:)' \quad \cdots \quad \mathbf{V}_1(:)' \quad \mathbf{D}_1(:)']$$

其中，$\mathbf{A}_N$是第N个分解等级的近似系数矩阵，$\mathbf{H}_i$, $\mathbf{V}_i$和$\mathbf{D}_i$（其中$i = 1, 2, \cdots, N$）是等级i的行、列和对角线的变换系数矩阵。这里，$\mathbf{H}_i(:)'$是通过级联矩阵$\mathbf{H}_i$的转置列形成的行向量。换言之，若

$$\mathbf{H}_i = \begin{bmatrix} 3 & -2 \\ 1 & 6 \end{bmatrix}$$

则

$$\mathbf{H}_i(:) = \begin{bmatrix} 3 \\ 1 \\ -2 \\ 6 \end{bmatrix}, \qquad \mathbf{H}_i(:)' = [3 \quad 1 \quad -2 \quad 6]$$

因为对c而言，方程假设为N分解（或是经过图7.2中的滤波器组），所以c包括$3N+1$个部分，即一个近似系数和N组行、列、对角线细节部分。注意，当$i = 1$时，可计算出最高尺度的系数；当$i = N$时，可计算出最低尺度的系数。于是，c的系数按尺度由低到高排序。

分解结构的矩阵S是一个大小为$(N+2) \times 2$的记录数组，其形式为

$$\mathbf{S} = [\mathbf{sa}_N; \quad \mathbf{sd}_N; \quad \mathbf{sd}_{N-1}; \quad \cdots \quad \mathbf{sd}_i; \quad \cdots \quad \mathbf{sd}_1; \quad \mathbf{sf}]$$

其中，$\mathbf{sa}_N$, $\mathbf{sd}_i$和$\mathbf{sf}$是大小为1×2的向量，它们分别包含第N级的近似矩阵$\mathbf{A}_N$的水平和垂直维数、第i级细节（即$i = 1, 2, \cdots, N$时的$\mathbf{H}_i$, $\mathbf{V}_i$和$\mathbf{D}_i$值）和原图像F。S中的信息可用于定位c中的单个近似和细节系数。注意，前述方程中的分号表示S的元素是以列向量的形式来组织的。

例7.4 用于执行变换分解向量c的小波工具箱函数

小波工具箱提供了多种函数，这些函数作为分解等级函数，用于定位、提取、重新格式化和/或处理c的近似矩阵及水平、垂直、对角线系数。在这里引入它们的目的是解释刚刚讨论的概念，并且为下一节将要介绍的函数做些准备。例如，考虑下列命令：

```
>> f = magic(8);
>> [c1, s1] = wavedec2(f, 3, 'haar');
>> size(c1)
ans =
     1    64
>> s1
s1 =
     1     1
     1     1
     2     2
     4     4
     8     8
```

```
>> approx = appcoef2(c1, s1, 'haar')
approx =
    260.0000

>> horizdet2 = detcoef2('h', c1, s1, 2)
horizdet2 =
    1.0e-013 *

             0   -0.2842
             0         0

>> newc1 = wthcoef2('h', c1, s1, 2);
>> newhorizdet2 = detcoef2('h', newc1, s1, 2)
newhorizdet2 =
     0       0
     0       0
```

其中，关于Haar小波的3级分解是使用函数 wavedec2 在一个大小为 8×8 的魔方图上执行的。得到的系数向量 c1 的大小是 1×64。由于 s1 的大小为 5×2，所以 c1 的系数范围为 $(N-2)=(5-2)=3$ 分解级别。因此，它会级联 $3N+1=3(3)+1=10$ 个近似和细节子矩阵的元素。基于 s1，这些子矩阵包括：(a)分解级别3的一个大小为 1×1 的近似矩阵和一个大小为 1×1 的细节矩阵[见 s1(1,:)和 s1(2,:)]；(b)分解级别2的3个大小为 2×2 的细节矩阵[见 s1(3,:)]；(c)分解级别1的3个大小为 4×4 的细节矩阵[见 s1(4,:)]。s1 的第5行包含了原图像 f 的尺寸。

矩阵 approx = 260 是使用工具箱函数 appcoef2 从 c1 中按如下语法提取的：

```
a = appcoef2(c, s, wname)
```

其中，wname 是来自于表7.1的一个小波名，a 是一个返回的近似矩阵。在第2级的水平细节系数是使用函数 detcoef2 得到的，该函数的语法为

```
d = detcoef2(o, c, s, n)
```

其中，对于水平、垂直和对角线的细节，o 设置为 'h'，'v' 或 'd'，n 是希望的分解等级。在本例中，返回了一个大小为 2×2 的矩阵 horizdet2。c1 中相应于 horizdet2 的系数接着由函数 wthcoef2 赋值为零，小波阈值函数 wthcoef2 的语法为

```
nc = wthcoef2(type, c, s, n, t, sorh)
```

其中，type 设置为阈值近似系数 'a'，而 'h'，'v' 或 'd' 分别设置水平、垂直、对角线细节的阈值。输入 n 是分解等级的一个向量，该向将基于向量 t 中的相应阈值进行阈值处理，而 sorh 针对于软硬阈值处理分别设置为 's' 或 'h'。若省略 t，则所有满足 type 和 n 规范的系数都将赋值为零。输出 nc 是已修改过的分解向量（即经阈值处理后的分解向量）。前面提到的所有三个小波工具箱函数还有其他语法形式，我们可以使用MATLAB的 help 命令来了解。

7.3.1 不使用小波工具箱编辑小波分解系数

若不使用小波工具箱，则记录矩阵S将是分别访问多尺度向量c的近似和细节系数的关键。在本节中，我们将使用S来建立一组处理c的通用程序。函数 wavework 是程序开发的基础，该函数是以字处理应用的剪切－复制－粘贴的方式为基础的。

```
function [varargout] = wavework(opcode, type, c, s, n, x)
%WAVEWORK is used to edit wavelet decomposition structures.
```

```
switch lower(type(1))          % Make pointers into C.
case 'a'
   nindex = 1;
   start = 1;    stop = elements(1);    ntst = nmax;
case {'h', 'v', 'd'}
   switch type
   case 'h', offset = 0;      % Offset to details.
   case 'v', offset = 1;
   case 'd', offset = 2;
   end
   nindex = size(s, 1) - n;      % Index to detail info.
   start = elements(1) + 3 * sum(elements(2:nmax - n + 1)) + ...
           offset * elements(nindex) + 1;
   stop = start + elements(nindex) - 1;
   ntst = n;
otherwise
   error('TYPE must begin with "a", "h", "v", or "d".');
end

switch lower(opcode)         % Do requested action.
case {'copy', 'cut'}
   y = repmat(0, s(nindex, :));
   y(:) = c(start:stop); nc = c;
   if strcmp(lower(opcode(1:3)), 'cut')
      nc(start: stop) = 0; varargout = {nc, y};
   else
       varargout = {y};
   end
case 'paste'
   if prod(size(x)) ~= elements(end - ntst)
      error('X is not sized for the requested paste.');
   else
      nc = c;   nc(start:stop) = x(:);     varargout = {nc};
   end
otherwise
   error('Unrecognized OPCODE.');
end
```

正如函数 wavework 检测其输入参数的合理性那样，c 的每个系数子矩阵中的元素数是通过 elements = prod(s,2) 计算出来的。回忆 3.4.2 节中的 MATLAB 函数 Y = prod(X,DIM) 沿维数 DIM 计算 X 的元素的积。第一个 switch 语句开始计算指向输入参数 type 和 n 的相关系数的一对指针。在近似值的情况下（即 case 'a'），计算很简单，因为系数总是在 c 的开始点（所以指针值从 1 开始）；结束索引，即指针 stop，是近似矩阵中的元素的数量，即 elements(1)。然而，在求一个细节系数子矩阵时，start 是通过求大于 n 的所有分解等级中的元素数目与 offset *elements(nindex) 的和来计算的；其中，offset 根据水平、垂直、对角线系数分别取 0，1 或 2，而 nindex 是一个对应于输入参数 n 的 s 的行的指针。

函数 wavework 中的第二个 switch 语句执行 opcode 要求的操作。在 'cut' 和 'copy' 的情况下，start 和 stop 之间的 c 的系数复制到了 y 中，而 y 是一个已预分配好的二维矩阵，其大小由 s 决定。上述操作是由语句 y = repmat(0,s(nindex,:)) 完成的，其中，MATLAB 的“复制矩阵”函数 B = repmat(A,M,N) 用于创建大矩阵 B，而 B 是由 A 组成的大小为 M × N 的矩阵。

在'paste'的情况下，x的元素复制到了nc中，nc为start和stop之间的输入c的副本。'cut'和'paste'操作都将返回一个新的分解向量nc。

下面三个函数（wavecut，wavecopy和wavepaste）使用函数wavework来处理c，所使用的语法如下所示：

```
function [nc, y] = wavecut(type, c, s, n)
%WAVECUT zeroes coefficients in a wavelet decomposition structure.
%   [NC, Y] = WAVECUT(TYPE, C, S, N) returns a new decomposition
%   vector whose detail or approximation coefficients (based on TYPE
%   and N) have been zeroed. The coefficients that were zeroed are
%   returned in Y.
%
%   INPUTS:
%     TYPE        Coefficient category
%     -------------------------------------------------------------------
%     'a'         Approximation coefficients
%     'h'         Horizontal details
%     'v'         Vertical details
%     'd'         Diagonal details
%
%     [C, S] is a wavelet data structure.
%     N specifies a decomposition level (ignored if TYPE = 'a').
%
%   See also WAVEWORK, WAVECOPY, and WAVEPASTE.

error(nargchk(3, 4, nargin));
if nargin == 4
   [nc, y] = wavework('cut', type, c, s, n);
else
   [nc, y] = wavework('cut', type, c, s);
end

function y = wavecopy(type, c, s, n)
%WAVECOPY Fetches coefficients of a wavelet decomposition structure.
%   Y = WAVECOPY(TYPE, C, S, N) returns a coefficient array based on
%   TYPE and N.
%
%   INPUTS:
%     TYPE        Coefficient category
%     -------------------------------------------------------------------
%     'a'         Approximation coefficients
%     'h'         Horizontal details
%     'v'         Vertical details
%     'd'         Diagonal details
%
%   [C, S] is a wavelet data structure.
%   N specifies a decomposition level (ignored if TYPE = 'a').
%
% See also WAVEWORK, WAVECUT, and WAVEPASTE.

error(nargchk(3, 4, nargin));
if nargin == 4
    y = wavework('copy', type, c, s, n);
```

```
else
    y = wavework('copy', type, c, s);
end

function nc = wavepaste(type, c, s, n, x)
%WAVEPASTE puts coefficients in a wavelet decomposition structure.
%   NC = WAVEPASTE(TYPE, C, S, N, X) returns the new decomposition
%   structure after pasting X into it based on TYPE and N.
%
%   INPUTS:
%     TYPE      Coefficient category
%     -------------------------------------------------------------------
%     'a'       Approximation coefficients
%     'h'       Horizontal details
%     'v'       Vertical details
%     'd'       Diagonal details
%
%     [C, S] is a wavelet data structure.
%     N specifies a decomposition level (Ignored if TYPE = 'a').
%     X is a two-dimensional approximation or detail coefficient
%       matrix whose dimensions are appropriate for decomposition
%       level N.
%
%   See also WAVEWORK, WAVECUT, and WAVECOPY.

error(nargchk(5, 5, nargin))
nc = wavework('paste', type, c, s, n, x);
```

例 7.5 使用函数 wavecut 和函数 wavecopy 处理 **c**

函数 wavecut 和 wavecopy 可以基于例 7.4 的结果重新产生小波工具箱：

```
>> f = magic(8);
>> [c1, s1] = wavedec2(f, 3, 'haar');
>> approx = wavecopy('a', c1, s1)
approx =
    260.0000

>> horizdet2 = wavecopy('h', c1, s1, 2)
horizdet2 =
    1.0e-013 *
           0   -0.2842
           0         0

>> [newc1, horizdet2] = wavecut('h', c1, s1, 2);
>> newhorizdet2 = wavecopy('h', newc1, s1, 2)
newhorizdet2 =
    0       0
    0       0
```

注意，所有提取的矩阵和前一个例子中的矩阵都是一样的。

7.3.2 显示小波分解系数

正如在 7.3 节开头指出的那样，系数被打包为一维小波分解向量 **c**。事实上，二维输出数组的系数来自于图 7.2 中的滤波器组。滤波器组的每次迭代会产生 4 个 1/4 大小的系数数组（忽略卷积过程导致的任何扩充）。

```
error(nargchk(2, 4, nargin));

if (ndims(c) ~= 2) | (size(c,1) ~= 1)
   error('C must be a row vector.'); end

if (ndims(s) ~= 2) | ~isreal(s) | ~isnumeric(s) | (size(s, 2) ~= 2)
   error('S must be a real, numeric two-column array.'); end

elements = prod(s,2);
if (length(c) < elements(end)) | ...
      ~(elements(1) + 3 * sum(elements(2: end-1)) >= elements(end))
   error(['[C S] must be a standard wavelet ' ...
          'decomposition structure.']);
end

if (nargin > 2) & (~isreal(scale) | ~isnumeric(scale))
   error('SCALE must be a real, numeric scalar.');
end

if (nargin > 3) & (~ischar(border))
   error('BORDER must be character string.');
end

if nargin == 2
   scale = 1;  % Default scale.
end

if nargin < 4
   border = 'absorb';  % Default border.
end

% Scale coefficients and determine pad fill.
absflag = scale < 0;
scale = abs(scale);
if scale == 0
   scale = 1;
end

[cd, w] = wavecut('a', c, s); w = mat2gray(w);
cdx = max(abs(cd(:))) / scale;
if absflag
   cd = mat2gray(abs(cd), [0, cdx]); fill = 0;
else
   cd = mat2gray(cd, [-cdx, cdx]); fill = 0.5;
end

% Build gray image one decomposition at a time.
for i = size(s, 1) - 2:-1:1
   ws = size(w);

   h = wavecopy('h', cd, s, i);
   pad = ws - size(h);     frontporch = round(pad / 2);
   h = padarray(h, frontporch, fill, 'pre');
   h = padarray(h, pad - frontporch, fill, 'post');
   v = wavecopy('v', cd, s, i);
   pad = ws - size(v);     frontporch = round(pad / 2);
   v = padarray(v, frontporch, fill, 'pre');
   v = padarray(v, pad - frontporch, fill, 'post');
   d = wavecopy('d', cd, s, i);
```

```
    pad = ws - size(d);      frontporch = round(pad / 2);
    d = padarray(d, frontporch, fill, 'pre');
    d = padarray(d, pad - frontporch, fill, 'post');

    % Add 1 pixel white border.
    switch lower(border)
    case 'append'
       w = padarray(w, [1 1], 1, 'post');
       h = padarray(h, [1 0], 1, 'post');
       v = padarray(v, [0 1], 1, 'post');
    case 'absorb'
       w(:, end) = 1;     w(end, :) = 1;
       h(end, :) = 1;     v(:, end) = 1;
    otherwise
       error('Unrecognized BORDER parameter.');
    end
    w = [w h; v d];                        % Concatenate coefs.
end

if nargout == 0
    imshow(w);                             % Display result.
end
```

“帮助文档”或函数wave2gray开头的部分描述了生成的输出图像w的结构。例如，w的左上角的子图像是由最后的分解步骤得到的近似数组，它被使用同样的分解步骤所生成的水平、对角线和垂直细节系数顺时针包围。得到的子图像发2 × 2数组接着被前一分解步骤的细节系数围绕（同样以顺时针方式）；而且，这种模式会一直持续到将所有的分解向量c的尺度附加到二维矩阵w。

刚刚提到的合成发生在函数wave2gray的惟一for循环内。在检查过输入的一致性后，将调用函数wavecut，以便从分解向量c中删除近似系数。这些系数接着利用函数mat2gray来缩放，以便于稍后的显示。修改后的分解向量cd（即无近似系数的c）也进行类似的缩放。对于输入scale为正值的情况，细节系数也被缩放，以便0值系数以中等灰度显示。对于所有必需的填充，fill的值为0.5（中等灰度）。若scale为负，则细节系数的绝对值将以相应于黑色的0值显示，且fill的值设置为0。在为显示缩放近似和细节系数之后，for循环的第一次迭代从cd中提取出最后一次分解的系数，并通过语顺w=[w h;v d]将它们附加到w中（在通过填充使得4个子图像的维数匹配并插入一个像素的白色边界之后）。该过程接着对c中的每个尺度重复进行。注意，函数wavecopy用来提取各种需要的细节系数以便形成w。

例7.6 使用函数wave2gray显示变换系数

下列命令用于计算图7.4中的图像关于四阶Daubechies小波的2尺度DWT，并显示得到的系数：

```
>> f = imread('vase.tif');
>> [c, s] = wavefast(f, 2, 'db4');
>> wave2gray(c, s);
>> figure; wave2gray(c, s, 8);
>> figure; wave2gray(c, s, -8);
```

最后三行语句生成的图像分别显示在图7.5(a)至图7.5(c)中。由于无其他缩放，所以细节系数的差别在图7.5(a)中几乎看不出来。而在图7.5(b)中，差别放大了8倍。注意沿着第1级系数子图像的边界的中等灰色填充；插入它的目的是为了使变换系数子图像间的维数变化一致。图7.5(c)显示了采用细节的绝对值后的效果。这里，所有的填充均为黑色。

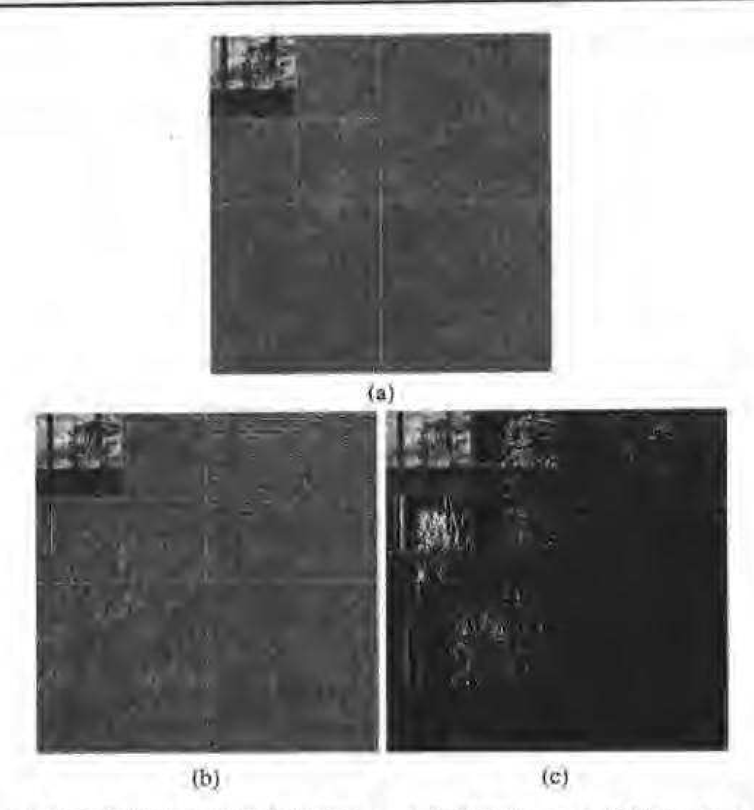


图 7.5 显示图 7.4 所示图像的 2 尺度小波变换：(a)自动缩放；(b)8 倍放大；(c)8 倍放大的绝对值

7.4 快速小波反变换

就像快速小波变换一样，快速小波反变换可以通过迭代地使用数字滤波器来计算。图 7.6 显示的是所要求的合成或重构滤波器组，该滤波器组会颠倒图 7.2 所示的分析或分解滤波器组的步骤。在每次迭代中，都对 4 个尺度 j 近似和细节子图像上取样（通过在每两个元素间插入零），并通过两个一维滤波器（一个执行子图像的列操作，另一个执行行操作）进行卷积操作。结果相加产生了尺度 $j+1$ 近似，这个过程会一直重复，直到原图像被重构为止。在卷积中应用的滤波器是一个在正变换中使用的小波函数。回忆可知，使用 7.2 节中的函数 `wfilters` 和 `wavefilter`（输入参数 `type` 设置为 `'r'` 表示"重构"）可得到上述滤波器。

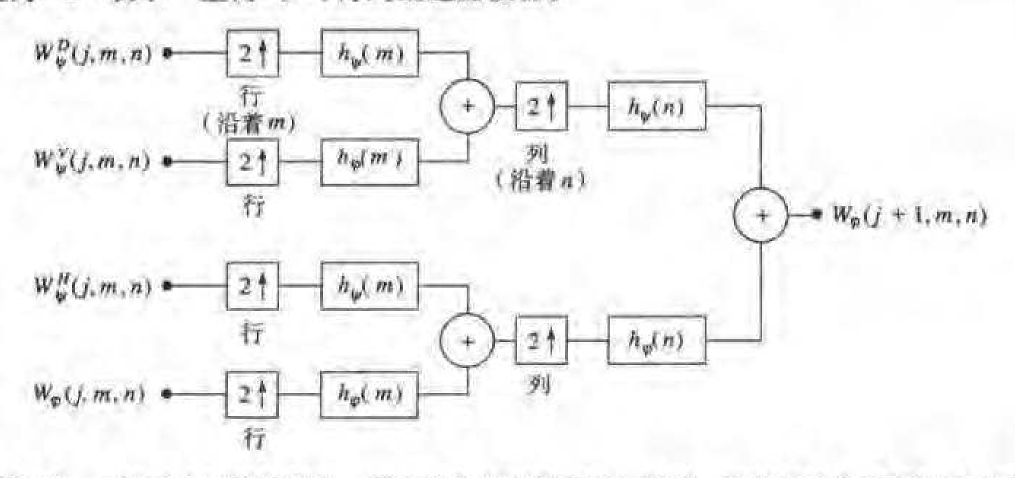


图 7.6 二维 FWT^{-1} 滤波器组。带上箭头的方框表示上取样，即在每两个元素间插入零

在使用小波工具箱时，函数 `waverec2` 用来计算小波分解结构 [C, S] 的反 FWT。该函数的调用语法为

g = waverec2(C, S, wname)

其中，g是重构后的二维图像（double类）。所需的重构滤波器也可通过语法

g = waverec2(C, S, Lo_R, Hi_R)

得到。

当小波工具箱不可用时，可以使用下面的自定义程序，在该程序中，我们调了函数waveback。该函数是无小波工具箱时，结合使用基于小波的软件包和IPT来进行图像处理所需要的最后一个函数。

```
function [varargout] = waveback(c, s, varargin)
%WAVEBACK Performs a multi-level two-dimensional inverse FWT.
%   [VARARGOUT] = WAVEBACK(C, S, VARARGIN) computes a 2D N-level
%   partial or complete wavelet reconstruction of decomposition
%   structure [C, S].
%
%   SYNTAX:
%     Y = WAVEBACK(C, S, 'WNAME');    Output inverse FWT matrix Y
%     Y = WAVEBACK(C, S, LR, HR);     using lowpass and highpass
%                                     reoonstruction filters (LR and
%                                     HR) or filters obtained by
%                                     calling WAVEFILTER with 'WNAME'.
%
%     [NC, NS] = WAVEBACK(C, S, 'WNAME', N);   Output new wavelet
%     [NC, NS] = WAVEBACK(C, S, LR, HR, N);    decomposition structure
%                                              [NC, NS] after N step
%                                              reconstruction.
%
%      See also WAVEFAST and WAVEFILTER.
% Check the input and output arguments for reasonableness.
error(nargohk(3, 5, nargin));
error(nargchk(1, 2, nargout));

if (ndims(c) ~= 2) | (size(c, 1) ~= 1)
   error('C must be a row vector.');
end

if (ndims(s) ~= 2) | ~isreal(s) | ~isnumeric(s) | (size(s, 2) ~= 2)
   error('S must be a real, numeric two-column array.');
end

elements = prod(s,2);
if (length(c) < elements(end)) | ...
      ~(elements(1) + 3 * sum(elements(2:end - 1)) >= elements(end))
   error(['[C S] must be a standard wavelet' ...
          'decomposition structure.']);
end

% Maximum levels in [C, S].
nmax = size(s, 1) - 2;
% Get third input parameter and Init check flags.
wname = varargin{1}; filterchk = 0; nchk = 0;

switch nargin
case 3
   if ischar(wname)
```

```
         [lp, hp] = wavefilter(wname, 'r'); n = nmax;
      else
         error('Undefined filter.');
      end
      if nargout ~= 1
         error('Wrong number of output arguments.');
      end
   case 4
      if ischar(wname)
         [lp, hp] = wavefilter(wname, 'r');
         n = varargin{2};   nchk = 1;
      else
         lp = varargin{1}; hp = varargin{2};
         filterchk = 1;     n = nmax;
         if nargout ~= 1
            error('Wrong number of output arguments.');
         end
      end
   case 5
      lp = varargin{1};     hp = varargin{2};     filterchk = 1;
      n = varargin{3};      nchk = 1;
   otherwise
      error('Improper number of input arguments.');
   end

   fl = length(lp);
   if filterchk                                         % Check filters.
      if (ndims(lp) ~= 2) | ~isreal(lp) | ~isnumeric(lp) ...
            | (ndims(hp) ~= 2) | ~isreal(hp) | ~isnumeric (hp) ...
            | (fl ~= length(hp)) | rem(fl, 2) ~= 0
         error(['LP and HP must be even and equal length real, ' ...
                'numeric filter vectors.']);
      end
   end

   if nchk & (~isnumeric(n) | ~isreal(n))          % Check scale N.
      error('N must be a real numeric.');
   end
   if (n > nmax) | (n < 1)
       error('Invalid number (N) of reconstructions requested.');
   end
   if (n ~= nmax) & (nargout ~= 2)
       error('Not enough output arguments.');
   end

   nc = c;    ns = s;    nnmax = nmax;                 % Init decomposition.
   for i = 1:n
      % Compute a new approximation.
      a = symconvup(wavecopy('a', nc, ns), lp, lp, fl, ns(3, :)) + ...
          symconvup(wavecopy('h', nc, ns, nnmax), ...
                    hp, lp, fl, ns(3, :)) + ...
          symconvup(wavecopy ('v', nc, ns, nnmax), ...
                    lp, hp, fl, ns(3, :)) + ...
          symconvup(wavecopy('d', nc, ns, nnmax), ...
                    hp, hp, fl, ns(3, :));
```

```
    % Update decomposition.
    nc = nc(4 * prod(ns(1,:)) + 1: end);      nc = [a(:)' nc];
    ns = ns(3:end, :);                          ns = [ns(1, :); ns];
    nnmax = size(ns, 1) - 2;
end

% For complete reconstructions, reformat output as 2-D.
if nargout == 1
    a = nc;    nc = repmat(0, ns(1, :));    nc(:) = a;
end

varargout{1} = nc;
if nargout == 2
    varagout{2} = ns;
end
%-------------------------------------------------------------------%
function z = symconvup(x, f1, f2, fln, keep)
% Upsample rows and convolve columns with f1; upsample columns and
% convolve rows with f2; then extract center assuming symmetrical
% extension.

y = zeros([2 1] .* size(x));       y(1:2:end, :) = x;
y = conv2(y, f1');
z = zeros([1 2] .* size(y));       z(:, 1:2:end) = y;
z = conv2(z, f2);
z = z(fln - 1:fln + keep(1) - 2, fln - 1:fln + keep(2) - 2);
```

函数waveback的主程序是一个简单的for循环，该循环根据期望的重构中的分解级数（即scales）进行迭代。正如我们所见，每个循环调用4次内部函数symconvup，并求返回矩阵的和。最初设定为c分解向量nc通过使用新建立的近似值a来替换传递给函数symconvup的4个系数矩阵，可不断地迭代更新。然后相应地修改记录矩阵ns——现在在分解结构[nc,ns]中有一个更小的尺度。这种操作顺序与图7.6中指出的稍有不同，最上面的两个输入组合产生了

$$[W_\psi^D(j,m,n)\uparrow^{2m} * h_\psi(m) + W_\psi^V(j,m,n)\uparrow^{2m} * h_\varphi(m)]\uparrow^{2n} * h_\psi(n)$$

其中，$\uparrow^{2m}$和$\uparrow^{2n}$分别表示沿着m和n的上取样。函数waveback使用等价计算

$$[W_\psi^D(j,m,n)\uparrow^{2m} * h_\psi(m)]\uparrow^{2n} * h_\psi(n) + [W_\psi^V(j,m,n)\uparrow^{2m} * h_\varphi(m)]\uparrow^{2n} * h_\psi(n)$$

函数symconvup执行卷积和上取样操作，以便依照前述方程计算图7.6中的一个输入对输出$W_\varphi(j+1,m,n)$的贡献。输入x首先在行方向上上取样，以便产生y，y然后按列与滤波器f1进行卷积操作。替代y的输出接着在列方向上上取样，并逐行与f2进行卷积操作来产生z。最后，z的中心keep元素（最后的卷积）将返回为新近似的输入x。

例7.7 比较函数waveback和函数waverec2的执行时间

通过简单地修改例7.3中的测试函数，下面的测试程序可比较小波工具箱函数waverec2和自定义函数waveback的执行时间：

```
function [ratio, maxdiff] = ifwtcompare(f, n, wname)
%IFWTCOMPARE Compare waverec2 and waveback.
%   [RATIO, MAXDIFF] = IFWTCOMPARE(F, N, WNAME) compares the
%   operation of Wavelet Toolbox function WAVEREC2 and custom function
%   WAVEBACK.
```

例 7.8　小波的方向性和边缘检测

考虑图 7.7(a)中大小为 500 × 500 的测试图像。在第 4 章中，为说明傅里叶变换的平滑及锐化特性，我们使用了这幅图像。这里，我们使用它来演示二维小波变换的方向敏感性和边缘检测的有效性：

```
>> f = imread('A.tif');
>> imshow(f);
>> [c, s] = wavefast(f, 1, 'sym4');
>> figure; wave2gray(c, s, -6);
>> [nc, y] = wavecut('a', c, s);
>> figure; wave2gray(nc, s, -6);
>> edges = abs(waveback(nc, s, 'sym4'));
>> figure; imshow(mat2gray(edges));
```

关于 'sym4' 小波，图 7.7(a)的单尺度小波变换的水平、垂直、对角线的方向性在图 7.7(b)中可清楚地看到。注意，原图像的水平边缘出现在图 7.7(b)的右上象限的水平细节系数中。对于图像的垂直边缘，可以在左下象限的垂直细节系数中类似地确定。要将这些信息合并成为一幅边缘图像，我们可以简单地把生成的变换的近似系数设为零，计算它的反变换，再对其取绝对值。修改过的变换和得到的边缘图像分别显示在图 7.7(c)和图 7.7(d)中。类似的过程可用于隔离垂直边缘或水平边缘。

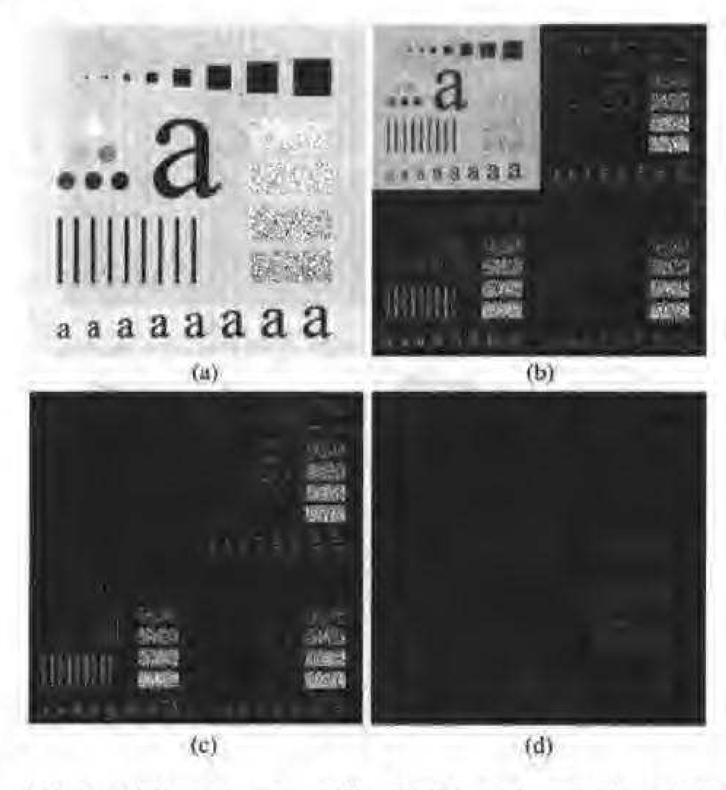


图 7.7　边缘检测中的小波：(a)一幅简单的测试图像；(b)它的小波变换；(c)将所有近似系数设置为零的变换；(d)计算反变换的绝对值所得到的边缘图像

例 7.9　基于小波的图像平滑或模糊

与傅里叶相对应的小波是平滑或模糊图像的有效手段。再次考虑图 7.7(a)所示的测试图像，它再次重复出现在图 7.8(a)中。图像的第四阶 symlets 小波变换示于图 7.8(b)中，很明显，这里已执行了 4 尺度分解。为了进行流线型的平滑处理，我们引入如下函数：

```
function [nc, g8] = wavezero(c, s, l, wname)
%WAVEZERO Zeros wavelet transform detail coefficients.
%   [NC, G8] = WAVEZERO(C, S, L, WNAME) zeros the level L detail
%   coefficients in wavelet decomposition structure [C, S] and
%   computes the resulting inverse transform with respect to WNAME
%   wavelets.

[nc, foo] = wavecut('h', c, s, l);
[nc, foo] = wavecut('v', nc, s, l);
[nc, foo] = wavecut('d', nc, s, l);
i = waveback(nc, s, wname);
g8 = im2uint8(mat2gray(i));
figure; imshow(g8);
```

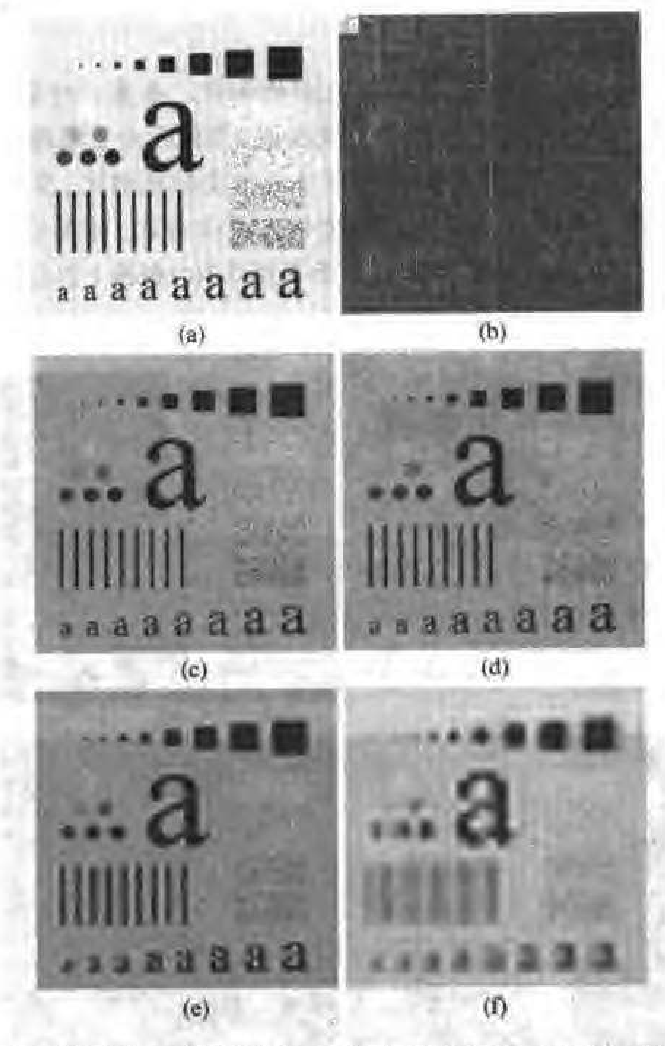


图 7.8 基于小波的图像平滑：(a)测试图像；(b)它的小波变换；(c)将第一级的细节系数设置为零后的反变换；(d)~(f)类似地将第二级、第三级、第四级的细节系数设置为零后的结果

使用 wavezero 函数，通过下列命令可生成图 7.8(a)的一系列越来越平滑的结果：

```
>> f = imread('A.tif');
>> [c, s]= wavefast(f, 4, 'sym4');
>> wave2gray(c, s, 20);
>> [c, g8] = wavezero(c, s, 1, 'sym4');
```

```
>> [c, g8] = wavezero(c, s, 2, 'sym4');
>> [c, g8] = wavezero(c, s, 3, 'sym4');
>> [c, g8] = wavezero(c, s, 4, 'sym4');
```

注意，图7.8(c)平滑后的图像仅有些许模糊，因为它只是通过将原图像的小波变换的第一级细节系数设为零（并计算修改后的变换的反变换）而得到的。图7.8(d)中的模糊现象加重了，因为它是将第二级细节系数设为零的效果。系数置零的处理在图7.8(e)中得以继续，这里第三级的细节系数也设为零；而对于图7.8(f)，所有的细节系数则都已被消除。图7.8(c)到图7.8(f)逐渐增加的模糊效果会使人联想起傅里叶变换中的类似结果。上述内容证明了小波域中的尺度和傅里叶域中的频率之间的内在联系。

例7.10 渐进重构

下面我们考虑图7.9(a)所示图像的4尺度小波变换的传输和重构，我们要浏览一个远程图像数据库来寻找一幅指定的图像。这里，我们将抛开本节开头描述的三步过程来考虑无傅里叶域对应技术的应用。数据库中的每幅图像都存储为一个多尺度的小波分解。这种结构非常适用于渐进的重构应用，尤其是用于存储变换的系数的一维分解向量假设采取7.3节中的通用格式时。对于此例中的4尺度变换，重构向量为

$$[\mathbf{A}_4(:)' \quad \mathbf{H}_4(:)' \quad \cdots \quad \mathbf{H}_i(:)' \quad \mathbf{V}_i(:)' \quad \mathbf{D}_i(:)' \quad \cdots \quad \mathbf{V}_1(:)' \quad \mathbf{D}_1(:)']$$

其中，$\mathbf{A}_4$是第四级分解的近似系数矩阵，而$\mathbf{H}_i$、$\mathbf{V}_i$和$\mathbf{D}_i$（$i=1,2,3,4$）分别是第i级的水平、垂直和对角线变换系数矩阵。若以从左到右的方法传送这个向量，则远程显示设备可以依据到达观测站的数据，逐渐建立最终的高分辨率图像的一个更高分辨率近似（基于用户的需要）。例如，当接收到系数$\mathbf{A}_4$时，可以看到图像的低分辨率版本［见图7.9(b)］。当接收到$\mathbf{H}_4$、$\mathbf{V}_4$和$\mathbf{D}_4$后，可构建一个更高分辨率的近似［见图7.9(c)］，依次类推。图7.9(d)至图7.9(f)给出了分辨率不断增加的重构结果。这种渐进的重构过程可用如下MATLAB命令来模拟：

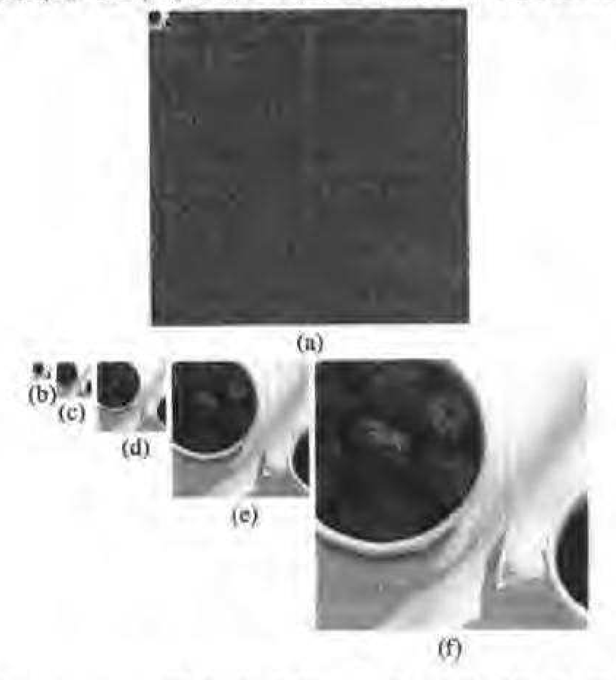


图7.9 渐进的重构：(a)一个4尺度小波变换；(b)左上角第四级近似图像；(c)集成第四级细节的一个精确近似；(d)~(f)集成更高级细节的分辨率的进一步改进

$$C_R = \frac{n_1}{n_2}$$

若压缩比为 10（或 10 : 1），则表明在压缩过的数据集中对于每 1 个单元，原图像有 10 个信息携带单元（如比特）。在 MATLAB 中，用于表示两幅图像文件和/或变量的比特数的比率可通过如下 M 函数计算出来：

```
function cr = imratio(f1, f2)
%IMRATIO Computes the ratio of the bytes in two images/variables.
%   CR = IMRATIO(F1, F2) returns the ratio of the number of bytes in
%   variables/files F1 and F2. If F1 and F2 are an original and
%   compressed image, respectively, CR is the compression ratio.

error(nargchk(2, 2, nargin));         % Check input arguments
cr = bytes(f1) / bytes(f2);           % Compute the ratio

%-------------------------------------------------------------------------%
function b = bytes(f)
% Return the number of bytes in input f. If f is a string, assume
% that it is an image filename; if not, it is an image variable.
if ischar(f)
   info = dir(f);      b = info.bytes;
elseif isstruct(f)
   % MATLAB's whos function reports an extra 124 bytes of memory
   % per structure field because of the way MATLAB stores
   % structures in memory. Don't count this extra memory; instead,
   % add up the memory associated with each field.
   b = 0;
   fields = fieldnames(f);
   for k = 1:length(fields)
      b = b + bytes(f.(fields{ k} ));
   end
else
   info = whos('f');      b = info.bytes;
end
```

例如，第 2 章中图 2.4(c)所示的 JPEG 编码图像的压缩可以通过以下命令进行计算：

```
>> r = imratio(imread('bubbles25.jpg'), 'bubbles25.jpg')
r =
    35.1612
```

注意，在函数 imratio 中，内部函数 b = bytes(f) 设计用于在(1)一个文件、(2)一个结构变量和/或(3)一个非结构变量中返回字节数。若 f 是一个非结构变量，则 2.2 节中引入的函数 whos 将用于得到它以字节为单位的尺寸。若 f 是一个文件名，则函数 dir 执行一个类似的服务程序。在所用的语法中，dir 返回一个带有域 name, date, bytes 和 isdir 的结构（关于结构的详细信息请参阅 2.10.6 节）。其中分别包括了文件名、修改状态、字节大小以及它是否为目录（若 isdir 为 1，则它是目录；若 isdir 为 0，则它不是目录）。最后，若 f 是一个结构，则 bytes 会递归地调用自身，以求出分配给该结构的每一个域的字节数之和。这就消除了与结构变量自身有关的开销（每个域 124 字节），而仅返回域中数据所需的字节数。函数 fieldnames 用于检索 f 中的一列域，语句

```
for k = 1:length(fields)
   b = b + bytes(f.(fields{ k} ));
```

执行递归调用。注意动态结构域名在递归调用 bytes 时的作用。若 S 是一个结构，且 F 是包含一个域名的串变量，则语句

```
S.(F) = foo;
field = S.(F);
```

采用动态结构域名语法分别来设置和/或得到结构域F的内容。

为了观察和/或使用压缩的（即编码后的）图像，必须把该图像送入一个解码器中（见图8.1），以便生成一个重构的输出图像$\hat{f}(x, y)$。一般而言，$\hat{f}(x, y)$可能是也可能不是$f(x, y)$的精确表示。若是，则称系统为无误差的、信息保持的或无损的；若不是，则在重构的图像中会有一度程度的失真。在称为有损压缩的后一种情况下，我们可以对x, y的任意值在$f(x, y)$和$\hat{f}(x, y)$之间定义误差$e(x, y)$，即

$$e(x, y) = \hat{f}(x, y) - f(x, y)$$

所以两幅图像间的总误差为

$$\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x, y) - f(x, y)]$$

同时，$f(x, y)$和$\hat{f}(x, y)$之间的均方根误差e_{rms}是$M \times N$数组的均方误差的平均值的平方根，或者

$$e_{rms} = \left[\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x, y) - f(x, y)]^2 \right]^{1/2}$$

下面的M函数可计算e_{rms}并在$e_{rms} \neq 0$时显示$e(x, y)$及其直方图。因为$e(x, y)$可以包含正负值，所以与仅处理图像数据的函数imhist相比，函数hist更常用于生成直方图。

```
function rmse = compare(f1, f2, scale)
%COMPARE Computes and displays the error between two matrices.
%   RMSE = COMPARE(F1, F2, SCALE) returns the root-mean-square error
%   between inputs F1 and F2, displays a histogram of the difference,
%   and displays a scaled difference image. When SCALE is omitted, a
%   scale factor of 1 is used.

% Check input arguments and set defaults.
error(nargchk(2, 3, nargin));
if nargin < 3
   scale = 1;
end

% Compute the root-mean-square error.
e = double(f1) - double(f2);
[m, n] = size(e);
rmse = sqrt(sum(e(:) .^ 2) / (m * n));

% Output error image & histogram if an error (i.e., rmse ~= 0).
if rmse
   % Form error histogram.
   emax = max(abs(e(:)));
   [h, x] = hist(e(:), emax);
   if length(h) >= 1
      figure; bar(x, h, 'k');

      % Scale the error image symmetrically and display
      emax = emax / scale;
      e = mat2gray(e, [-emax, emax]);
```

```
        figure; imshow(e);
    end
end
```

最后，我们注意到图8.1中的编码器负责减少输入图像的编码、像素间和/或心理视觉冗余。在编码处理的第一个阶段，映射变换器将输入图像变换成一种（通常不可见的）格式，以便减少像素间的冗余。在第二个阶段，量化器根据预定义的逼真度标准来减少映射变换器输出的精确性，以便试图去除心里视觉冗余数据。这种操作是不可逆的，当我们希望无误差压缩时，则必须将其忽略。在第三个即最后一个处理阶段，符号编码器根据所用的码字对量化器输出和映射输出创建码字（减少编码冗余）。

图8.1中的解码器仅包括两部分：一个符号解码器和一个反映射变换器。这些块以相反的顺序执行编码器的符号编码和映射变换器的逆操作。由于量化器不可逆，因此不包括反量化块。

8.2 编码冗余

令具有相关概率$p_r(r_k)$的离散随机变量r_k，$k = 1, 2, \cdots, L$表示一幅L级灰度图像的灰度级。正如在第3章中那样，r_1对应于灰度级0（因为MATLAB数组索引不能为0）且

$$p_r(r_k) = \frac{n_k}{n} \quad k = 1, 2, \cdots, L$$

其中n_k是图像中出现第k级灰度的次数，n是图像的总像素数。若用于表示r_k的每个值的比特数是$l(r_k)$，则用于表示每个像素的平均比特数为

$$L_{\text{avg}} = \sum_{k=1}^{L} l(r_k) p_r(r_k)$$

换言之，分配给各个灰度级值的码字的平均长度，就是用于表示每个灰度级的比特数与该灰度级出现的概率的乘积之和。这样，用于编码一幅大小为$M \times N$的图像所要求的总比特数就是MNL_{avg}。

当使用m比特的常用二进制码表示一幅图像的灰度级时，上述等式的右边减少到m比特。换言之，当m替代了$l(r_k)$时，$L_{\text{avg}} = m$。常数m可能在总和之外，对于$1 \leqslant k \leqslant L$，仅剩下$p_r(r_k)$之和，当然它一定等于1。如表8.1中所示例的那样，当图像的灰度级使用常用二进制码编码时，编码冗余几乎总是存在的。在该表中，一幅四灰度级图像采用了固定的和可变长度的编码，其灰度级的分布已在表中的第2列给出。第3列的2比特二进制码（Code 1）的平均长度为2。Code 2（第5列）所要求的平均比特数为

$$\begin{aligned} L_{\text{avg}} &= \sum_{k=1}^{4} l_2(k) p_r(r_k) \\ &= 3(0.1875) + 1(0.5) + 3(0.125) + 2(0.1875) = 1.8125 \end{aligned}$$

产生的压缩比为$C_r = 2/1.8125 \approx 1.103$。Code 2所实现的压缩的基础是其码字为变长的，即允许将最短的代码分配给图像中最常出现的灰度级。

表8.1 编码冗余的说明：Code 1中$L_{\text{avg}} = 2$，Code 2中$L_{\text{avg}} \approx 1.81$

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
r_1	0.1875	00	2	011	3
r_2	0.5000	01	2	1	1
r_3	0.1250	10	2	010	3
r_4	0.1875	11	2	00	2

人们经常提出的一个问题是，表示一幅图像的灰度级到底需要多少比特？换言之，是否存在在不丢失信息的条件下充分描绘一幅图像的最小数据量？信息论提供了回答这一相关问题的数学框架。它的基本前提是信息的产生可以通过概率过程来建立模型，这个过程可以利用与直觉相符的方式来度量。为与这个假定相一致，概率为$P(E)$的一个随机事件E包含了下面的单元信息：

$$I(E) = \log \frac{1}{P(E)} = -\log P(E)$$

若$P(E)=1$（即该事件总会发生），则$I(E)=0$且它没有信息。换言之，因为没有与这个事件相关的不确定因素，所以也就没有表示事件发生的、需要传递的信息。若在一组可能的离散事件$\{a_1, a_2, \cdots, a_J\}$中给定一个随机事件源，则与之相关的概率为$\{P(a_1), P(a_2), \cdots, P(a_J)\}$，且每个源输出的平均信息（称为源的熵）为

$$H = -\sum_{j=1}^{J} P(a_j) \log P(a_j)$$

若将一幅图像看成是一个发出"灰度级源"的样本，则我们可以利用被观测图像的灰度级直方图来模拟该源的符号概率，并生成该源的熵的一个估计$\widetilde{H}$（称为一阶估计）：

$$\widetilde{H} = -\sum_{k=1}^{L} p_r(r_k) \log p_r(r_k)$$

这样的估计可通过如下M函数计算出来，且在已对每个灰度级单独进行了编码的假设下，仅通过减少编码冗余就可以达到压缩的下界。

```
function h = entropy(x, n)
%ENTROPY Computes a first-order estimate of the entropy of a matrix.
%   H = ENTROPY(X, N) returns the first-order estimate of matrix X
%   with N symbols (N = 256 if omitted) in bits/symbol. The estimate
%   assumes a statistically independent source characterized by the
%   relative frequency of occurrence of the elements in X.

error(nargchk(1, 2, nargin));       % Check input arguments
if nargin < 2
   n = 256;                         % Default for n.
end

x = double(x);                      % Make input double
xh = hist(x(:), n);                 % Compute N-bin histogram
xh = xh / sum(xh(:));               % Compute probabilities

% Make mask to eliminate 0's since log2(0) = -inf.
i = find(xh);

h = - sum(xh(i) .* log2(xh(i)));    % Compute entropy
```

注意MATLAB函数`find`的运用，该函数用来决定直方图`xh`中的非零元素的索引。语句`find(x)`等价于`find(x ~= 0)`。函数`entropy`使用`find`来建立一个索引向量`i`，并将其输入到直方图`xh`中，通过最后一条语句中的熵计算，该直方图随后用于消除所有的零值元素。若未完成这项任务，则函数`log2`会在任意符号概率为0时，强制输出`h`为`NaN`（`0*-inf`不是一个数）。

例8.1　计算一阶熵估计

考虑一幅大小为4×4的简单图像，其直方图（见下面代码中的`p`）模拟表8.1中的符号概率。下面的命令行用于生成上述图像并计算其熵的一阶估计。

```
>> f = [119 123 168 119; 123 119 168 168];
>> f = [f; 119 119 107 119; 107 107 119 119]

f =
    119   123   168   119
    123   119   168   168
    119   119   107   119
    107   107   119   119

p = hist(f(:), 8);
p = p / sum(p)
p =
    0.1875   0.5   0.125   0   0   0   0   0.1875
h = entropy(f)

h =
    1.7806
```

在表 8.1 中，Code 2（$L_{avg} \approx 1.81$）近似一阶熵的估计，并且对于图像 f 来说是一个最小长度的二进制码。注意，灰度级 107 对应于 r_1 以及表 8.1 中的相应二进制码字 001_2，灰度级 119 对应于 r_2 及代码 1_2，而 123 和 168 分别对应于 010_2 和 00_2。

8.2.1 霍夫曼码

当对一幅图像的灰度级或一个灰度级映射操作的输出（像素差、游程长度等）进行编码时，在每次编码一个源符号的限制条件下，对于每个源符号（如灰度级值），霍夫曼码包含了最小可能的代码符号（即比特）数。

霍夫曼方法的第一步是，通过对符号的概率排序并将最低概率符号组合为下一次源约简时替代这些符号的单个符号，来建立一个源约简序列。图 8.2(a)示例了表 8.1 中的灰度级分布的这种过程。在最左侧，初始源符号集和它们的概率按照降序概率值由高到底排序。要形成第一次源约简，底部的两个概率值 0.125 和 0.1875 合并形成了概率值为 0.3125 的“复合符号”。这个复合符号及其相关的概率放置在第一个源约简列中，以便约简后的源的概率还是从最大到最小排序。这个过程一直重复到约简后的源只有两个符号为止（在最右侧）。

霍夫曼方法的第二步是对每个约简的源编码，编码从最小的源开始一直到原始源。两个符号的源的最短二进制码就是 0 和 1。如图 8.2(b)所示，这些符号分配给了右边的两个符号（这种分配是任意的；颠倒 0 和 1 的顺序也是可以的）。概率为 0.5 的约简源符号是通过合并约简源左侧的两个符号而生成的，用来编码的 0 现在分配给了这两个符号，而 0 和 1 已任意地附加给了每个符号，以便区分它们。这种操作接着对每个约简的源重复进行，直到到达原始源。最后的代码出现在图 8.2(b)的左侧（第 3 列）。

图 8.2(b)与表 8.1 中的霍夫曼码是瞬时的、惟一可解码的分组编码。之所以是分组编码，是因为每个源符号映射到了一个固定的码符号序列；之所以是瞬时的，是因为在码符号串中的每一个码字可以不参照随后的符号而被解码。换言之，在任意一个给定的霍夫曼码中，没有码字是其他码字的前缀。并且之所以是惟一可解码的，是因为码符号串仅有惟一的解码方法。这样，霍夫曼码的符号的任意串可以通过从左到右的方式检测单独的符号来解码。对于例 8.1 中的 4 × 4 图像，基于图 8.2(b)中的霍夫曼码由顶到底、从左到右进行编码，产生一个 29 比特的字符串 10101011010110110000011110011。因为我们应用的是一个瞬时的、惟一可解码的分组编码，所以不需要在已编码的像素间插入定界符。从左到右查看结果串可知，第一个有效的码字是 1，它是符

号a_2或119灰度级的编码。接下来的有效码字是010，它对应于灰度级123。继续使用这种方式，最终获得了一幅完整的解码后的图像，这幅图像与例子中的 f 相同。

原始源		源约简	
符号	概率	1	2
a_2	0.5	0.5	0.5
a_4	0.1875	0.3125	0.5
a_1	0.1875	0.1875	
a_3	0.125		

(a)

原始源			源约简			
符号	概率	码字	1		2	
a_2	0.5	1	0.5	1	0.5	1
a_4	0.1875	00	0.3125	01	0.5	0
a_1	0.1875	011	0.1875	00		
a_3	0.125	010				

(b)

图 8.2　霍夫曼码：(a)源约简；(b)码分配过程

上面讨论的源约简和码分配过程由下面的 M 函数 huffman 实现：

```
function CODE = huffman(p)
%HUFFMAN Builds a variable-length Huffman code for a symbol source.
%   CODE = HUFFMAN(P) returns a Huffman code as binary strings in
%   cell array CODE for input symbol probability vector P. Each word
%   in CODE corresponds to a symbol whose probability is at the
%   corresponding index of P.
%
%   Based on huffman5 by Sean Danaher, University of Northumbria,
%   Newcastle UK. Available at the MATLAB Central File Exchange:
%   Category General DSP in Signal Processing and Communications.

% Check the input arguments for reasonableness.
error(nargchk(1, 1, nargin));
if (ndims(p) ~= 2) | (min(size(p)) > 1) | ~isreal(p) | ~isnumeric(p)
   error('P must be a real numeric vector.');
end

% Global variable surviving all recursions of function 'makecode'
global CODE
CODE = cell(length(p), 1);     % Init the global cell array

if length(p) > 1               % When more than one symbol ...
   p = p / sum(p);             % Normalize the input probabilities
   s = reduce(p);              % Do Huffman source symbol reductions
   makecode(s, []);            % Recursively generate the code
else
   CODE = {'1'};               % Else, trivial one symbol case!
end;
%-------------------------------------------------------------------%
```

```
function s = reduce(p);
% Create a Huffman source reduction tree in a MATLAB cell structure
% by performing source symbol reductions until there are only two
% reduced symbols remaining

s = cell(length(p), 1);

% Generate a starting tree with symbol nodes 1, 2, 3, ... to
% reference the symbol probabilities.
for i = 1:length(p)
   s{i} = i;
end

while size(s) > 2
   [p, i] = sort(p);          % Sort the symbol probabilities
   p(2) = p(1) + p(2);        % Merge the 2 lowest probabilities
   p(1) = [];                 % and prune the lowest one

   s = s(i);                  % Reorder tree for new probabilities
   s{2} = {s{1}, s{2}};       % and merge & prune its nodes
s(1) = [];                    % to match the probabilities
end
%-------------------------------------------------------------------------%
function makecode(sc, codeword)
% Scan the nodes of a Huffman source reduction tree recursively to
% generate the indicated variable length code words.

% Global variable surviving all recursive calls
global CODE
if isa(sc, 'cell')                          % For cell array nodes,
   makecode(sc{1}, [codeword 0]);           % add a 0 if the 1st element
   makecode(sc{2}, [codeword 1]);           % or a 1 if the 2nd
else                                        % For leaf (numeric) nodes,
   CODE{sc} = char('0' + codeword);         % create a char code string
end
```

下面的命令行使用 huffman 来产生图 8.2 中的代码：

```
>> p = [0.1875 0.5 0.125 0.1875];
>> c = huffman(p)

c =
    '011'
    '1'
    '010'
    '00'
```

注意，输出是长度可变的字符数组，其中每一行是一串 0 和 1，对应于 p 中的索引符号的二进制码。例如，'010'（在数组索引 3 处）是概率为 0.125 的灰度级的代码。

在函数 huffman 的开始行上，已经对输入参量 p（将被编码的符号的输入符号概率向量）的合理性进行了检查，而且全局变量 CODE 已经用 length(p) 行和一个单列初始化为一个 MATLAB 单元数组（已在 2.10.6 节中定义）。所有的 MALAB 全局变量必须在函数中使用如下语句进行声明：

```
global X Y Z
```

该语句使变量X，Y，Z可用于已对其进行声明的函数中。当几个函数声明相同的全局变量时，它们共享该变量的单一副本。在函数huffman中，主程序和内部函数makecode共享全局变量CODE。注意，全局变量名一般为大写。非全局变量就是局部变量，只有在定义它们时才可以在函数中使用（不可以在其他函数或基本工作空间中使用）。局部变量一般用小写字母表示。

在函数huffman中，已使用cell函数初始化了CODE，函数cell的语法为

```
X = cell(m, n)
```

它产生一个可以按单元或内容引用的空矩阵的$m \times n$数组。圆括号"()"用于单元索引，大括号"{}"用于内容索引。因此，X(1) =[] 从单元数组中找到并删除元素1，X{1} = [] 则把第一个单元数组中的元素放到空矩阵中。换言之，X{1}指的是X中第一个元素（一个数组）的内容；而X(1)指的是元素本身（而不是它的内容）。由于单元数组可以嵌套在其他单元数组中，所以语法X{1}{2}指的是单元数组的第二个元素的内容，该内容在单元数组X的第一元素中。

在初始化CODE且（在p = p/sum(p)语句中）归一化输入概率向量后，就可在两个步骤中创建归一化概率向量p的霍夫曼码。由主程序的s = reduce(p)语句初始化的第一步是调用内部函数reduce，该函数的用途是执行图8.2(a)所示的源约简。在函数reduce中，一个初始为空的源约简单元数组s（大小与CODE匹配）中的元素被初始化为它们的索引。例如，s{1} = 1，s{2} = 2，等等。为了实现源约简，在while numel(s)>2循环中建立一个二叉树的单元等效流程。在循环的每一次迭代中，向量p以概率的升序排列。这可通过函数sort来完成，该函数的语法为

```
[y, i] = sort(x)
```

其中，输出y是x中经过排序的元素，索引向量i满足y = x(i)。当已对p排序后，合并最低的两个概率，并将复合概率放在p(2)中，同时把p(1)删除。然后，通过使用s = s(i)，重新排列源约简单元数组，以匹配基于索引向量i的p。最后，s{2}通过s{2} ={s{1}，s{2}}用一个包含合并的概率索引的二元素单元数组来代替（一个内容索引的例子），并采用单元索引来删除两个合并的元素的第一个元素s(1)，即令s(1) =[]。该过程一直重复进行，直到s中只有两个元素为止。

图8.3显示了表8.1和图8.2(a)中对符号概率进行处理的最后输出。图8.3(b)和图8.3(c)是通过在huffman主程序最后的两个语句间插入

```
celldisp(s);
cellplot(s);
```

产生的。MATLAB函数celldisp递归地打印一个单元数组的内容；函数cellplot产生类似嵌套逻辑单元的单元数组的图形描述。注意，图8.3(b)中的单元数组和图8.3(a)中的源约简树节点之间的一一对应关心：(1)树中的每两路分支（它代表一个源约简）与s中二元素的单元数组相对应；(2)每个二元素的单元数组都包含符号索引，该符号在相应的源约简中被合并。例如，合并树底部的a_1和a_3产生了二元素单元数组s{1}{2}，其中s{1}{2}{1} = 3，s{1}{2}{2} = 1（分别是符号a_3和a_1的索引）。树根是最高层的二元素单元数组s。

代码生成的最后一步（即基于源约简的单元数组s的代码分配）是通过调用函数hffman的最后一条语句makecode(s,[])进行的。该调用将启动基于图8.2(b)所示过程的递归代码分配操作。虽然递归方式一般并不提供存储（因为正在处理的值的堆栈必须存放在某些地方）或是提高速度，但它还是具有代码紧凑且易于理解的优点，特别是在处理已递归定义的数据结构（例如树）时。任何MATLAB函数都可以递归地使用，即这些函数可以直接地或间接地调用它自身。当使用递归方式时，每一个函数调用都可以产生一组不同于所有以前变量的新局部变量。

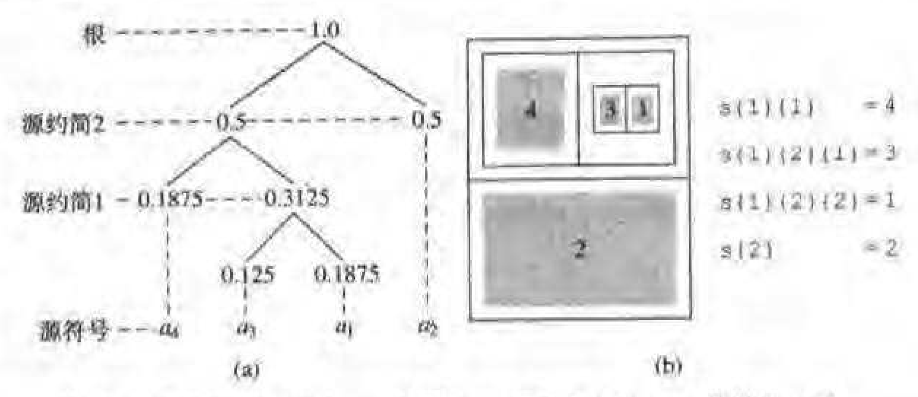


图 8.3　使用函数huffman对图8.2(a)进行源约简：(a)等效的二叉树；(b)cellplot(s)产生的显示；(c)celldisp(s)的输出

内部函数makecode接受两个输入：codeword，即一个元素为0和1的数组；sc，即一个源约简单元数组元素。当sc本身是一个单元数组时，它包括两个源符号（或复合符号），它们在源约简处理中被连接到一起。因为这些符号都必须单独进行编码，所以对元素执行一对递归调用（对makecode），同时还带有两个合适的更新码字（将一个0和1添加到输入codeword中）。当sc不包括单元数组时，它是原始源符号的索引，并且为其分配了一串由输入codeword通过CODE(sc) = char('0' + codeword)产生的二进制串。如同在2.10.5节中指出的那样，MATLAB函数char把一个包含代表字符码的正整数的数组转换为一个MATLAB字符数组（前127个码是ASCII码）。例如，char('0'+[0 1 0])产生字符串'010'，即在一个0的ASCII码上加0来产生一个ASCII码'0'，在一个ASCII码'0'上加1来产生1的ASCII码，也就是'1'。

表8.2详细列出了一系列makecode的调用，这些调用产生了图8.3中的源约简单元数组的相应结果。为了对4个源符号进行编码，需要7次调用。第一次调用（表8.2中的第一行）由huffman的主程序执行，并启动编码过程，同时输入codeword与sc分别设置为空矩阵和单元数组s。为了与MATLAB的标准表示法一致，{1 × 2 cell}表示一个有着一行和两列的单元数组。因为sc在第一次调用中几乎总是一个单元数组（单个符号源是一个例外），所以出现了两次递归调用（见表中的第2行和第7行）。这些调用的第一次调用引起了两次更多的调用（行3，行4），第二次调用则引起了其他的两次调用（行5，行6）。

表 8.2　图 8.3 中的源约简单元数组的码分配过程

调用	源程序	sc	码字
1	main routine	{1x2 cell} [2]	[]
2	makecode	[4] {1x2 cell}	0
3	makecode	4	0 0
4	makecode	[3] [1]	0 1
5	makecode	3	0 1 0
6	makecode	1	0 1 1
7	makecode	2	1

任何时候sc都不是一个单元数组，如表中的第3行、第5行、第6行和第7行所示，额外的递归也是不必要的；码串由codeword产生并分配到源符号，源符号的索引则作为sc进行传递。

8.2.2　霍夫曼编码

霍夫曼编码的生成不是压缩过程。为了实现构建在霍夫曼码中的压缩，对于产生代码的符号，不管它们是灰度级、游程长度或其他灰度映射操作的输出，都必须依照生成的代码对其进行变换或映射。

例8.2 MATLAB中的变长码映射

考虑大小为4 × 4的16字节图像：

```
>> f2 = uint8([2 3 4 2; 3 2 4 4; 2 2 1 2; 1 1 2 2])
f2 =
    2    3    4    2
    3    2    4    4
    2    2    1    2
    1    1    2    2
>> whos('f2')
    Name      Size      Bytes     Class
    f2        4x4       16        uint8 array
Grand total is 16 elements using 16 bytes
```

f2中的每一个像素都是一个8比特字节；16字节用于表示整个图像。因为f2的灰度级不是等概率的，所以一个变长码（像在前一节指出的那样）会减少表示图像所需要的存储量。函数huffman计算一个这样的码：

```
>> c = huffman(hist(double(f2(:)), 4))
c =
    '011'
    '1'
    '010'
    '00'
```

因为霍夫曼码与被编码的源符号出现的频率有关（不是信号本身），所以c就和为例8.1中的图像所构建的码相同。事实上，图像f2可以通过例8.1中的f来得到，方法是分别将灰度级107，119，123和168映射为1，2，3和4。对于任何一幅图像，p=[0.1875 0.5 0.125 0.1875]。基于码c来编码f2的一种简便方法是执行一个简单的查表操作：

```
>> h1f2 = c(f2(:))'
h1f2 =

    Columns 1 through 9
    '1'   '010'   '1'   '011'   '010'   '1'   '1'   '011'   '00'

    Columns 10 through 16
    '00'   '011'   '1'   '1'   '00'   '1'   '1'
>> whos('h1f2')
    Name      Size      Bytes     Class
    h1f2      1x16      1530      cell array
Grand total is 45 elements using 1530 bytes
```

其中，f2（一个UINT8类二维数组）被变换成了一个大小为1 × 16的单元数组h1f2。h1f2的元素是可变长度的串，这些串对应于f2中从上到下、从左到右的像素（顺时针）。正如我们所见到的一样，被编码的图像使用了1530字节的存储量，这几乎是f2要求的存储量的100倍。对h1f2使用单元数组是合理的，因为它是处理不同数据的数组的两个标准MATLAB数据结构之一（见2.10.6节）。在h1f2的情形下，不同之处在于字符串的长度，以及通过单元数组直接处理它所付出的代价是存储器的总开销（单元数组所固有的特性），它要求必须追踪变长元素的位置。通过把h1f2变换为一个二维字符数组，我们可消除这种开销：

注意，我们已用dec2bin来显示h3f2.code的各个比特。若忽略末尾的模16的填充比特（即最后的三个0），则32比特编码等同于先前产生的（见8.2.1节）、瞬时且惟一可解码的29比特分组码10101011010110110000011110011。

如同我们在前一个例子中说明的那样，函数mat2huff会将解码一个已编码输入数组所需要的信息（如其原始维数和符号概率）嵌入到一个单独的MATLAB结构变量中。该结构中的信息已在mat2huff的帮助文本中进行了说明：

```
function y = mat2huff(x)
%MAT2HUFF Huffman encodes a matrix.
%   Y = MAT2HUFF(X) Huffman encodes matrix X using symbol
%   probabilities in unit-width histogram bins between X's minimum
%   and maximum values. The encoded data is returned as a structure
%   Y:
%      Y.code   The Huffman-encoded values of X, stored in
%               a uint16 vector. The other fields of Y contain
%               additional decoding information, including:
%      Y.min    The minimum value of X plus 32768
%      Y.size   The size of X
%      Y.hist   The histogram of X
%
%   If X is logical, uint8, uint16, uint32, int8, int16, or double,
%   with integer values, it can be input directly to MAT2HUFF. The
%   minimum value of X must be representable as an int16.
%
%   If X is double with non-integer values---for example, an image
%   with values between 0 and 1---first scale X to an appropriate
%   integer range before the call. For example, use Y =
%   MAT2HUFF(255*X) for 256 gray level encoding.
%
%   NOTE: The number of Huffman code words is round(max(X(:))) -
%   round(min(X(:))) + 1. You may need to scale input X to generate
%   codes of reasonable length. The maximum row or column dimension
%   of X is 65535.
%
% See also HUFF2MAT.

if ndims(x) ~= 2 | ~isreal(x) | (~isnumeric(x) & ~islogical(x))
   error('X must be a 2-D real numeric or logical matrix');
end

% Store the size of input x.
y.size = uint32(size(x));

% Find the range of x values and store its minimum value biased
% by +32768 as a UINT16.
x = round(double(x));
xmin = min(x(:));
xmax = max(x(:));
pmin = double(int16(xmin));
```

```
pmin = uint16(pmin + 32768);    y.min = pmin;

% Compute the input histogram between xmin and xmax with unit
% width bins, scale to UINT16, and store.
x = x(:)';
h = histc(x, xmin:xmax);
if max(h) > 65535
   h = 65535 * h / max(h);
end
h = uint16(h); y.hist = h;

% Code the input matrix and store the result.
map = huffman(double(h));              % Make Huffman code map
hx = map(x(:) - xmin + 1);             % Map image
hx = char(hx)';                        % Convert to char array
hx = hx(:)';
hx(hx == ' ') = [];                    % Remove blanks
ysize = ceil(length(hx) / 16);         % Compute encoded size
hx16 = repmat('0', 1, ysize * 16);     % Pre-allocate modulo-16 vector
hx16(1:length(hx)) = hx;              % Make hx modulo-16 in length
hx16 = reshape(hx16, 16, ysize);       % Reshape to 16-character words
hx16 = hx16' - '0';                    % Convert binary string to decimal
twos = pow2(15:-1:0);
y.code = uint16(sum(hx16 .* twos(ones(ysize, 1), :), 2))';
```

注意语句 y = mat2huff(x)，霍夫曼方法利用 x 的最小值和最大值之间的单位宽度直方图 bin 对输入矩阵 x 进行编码。在以后对 y.code 中的编码数据进行解码时，对其解码的霍夫曼码必须通过 x 的最小值 y.min 和 x 的直方图 y.hist 来重建。替代保留霍夫曼码本身，mat2huff 保留的是需要用于重新生成它的概率信息。利用这些信息和存储在 y.size 中的 x 矩阵的原始维数，8.2.3 节中的函数 huff2mat 可用来解码 y.code，从而重构 x。

生成 y.code 的过程如下所示：

1. 使用 x 的最小值和最大值之间的单位宽度 bin 计算输入 x 的直方图 h，并缩放该直方图，以使其为一个 UINT16 向量。
2. 基于缩放的直方图 h，利用 huffman 产生霍夫曼码 map。
3. 利用 map 映射输入 x（这会产生一个单元数组）并把它转换成一个字符数组 hx，删除像在例 8.2 的 h2f2 中插入的空格。
4. 构建向量 hx，使其字符排列为 16 个字符的字符段。这种处理可以通过如下方式来完成：创建一个用来保存它的模 16 字符向量（代码中的 hx16），把 hx 的元素复制到该字符向量中，并且用数组 ysize 将其大小调整为 16 行，其中 ysize = ceil(length(hx)/16)。回忆 4.2 节可知，函数 ceil 会对一个趋向于正无穷的数四舍五入。通用的 MATLAB 函数

   ```
   y = reshape(x, m, n)
   ```

   会返回一个大小为 m × n 的矩阵，该矩阵的元素按列取自 x。若 x 没有 m*n 个元素，则会返回一个错误。
5. 将 hx16 的 16 字符元素转换为 16 比特二进制数（即 unit16 类数）。三条语句被更紧凑的语句 y = uint16(bin2dec(hx16')) 所取代。它们是 bin2dec 的核心，bin2dec 将返回一个二进制串的十进制数（如 bin2dec('101') 返回 5），但处理速度会更快。MATLAB 函数 pow2(y) 用于返回一个数组，该数组的元素是 2 的 y 次幂。换言之，twos = pow2(15:-1:0) 创建了数组[32768 16384 8192 ... 8 4 2 1]。

例8.3 使用函数 mat2huff 进行编码

为了进一步说明霍夫曼编码的压缩性能，我们现在来考虑图8.4(a)所示的一幅大小为512×512的8比特单色图像。该图像用使用函数 mat2huff 进行压缩，所用的命令如下：

```
>> f = imread('Tracy.tif');
>> c = mat2huff(f);
>> cr1 = imratio(f, c)

cr1 =
    1.2191
```



图8.4 大小为512×512的8比特单色图像及其右眼特写

通过删除与常规8比特二进制编码相关的编码冗余，图像已被压缩到其原始大小的80%左右（甚至包含了解码所需的开销信息）。

因为 mat2huff 的输出是一个结构，所以我们使用函数 save 将它写到磁盘中：

```
>> save SqueezeTracy c;
>> cr2 = imratio('Tracy.tif', 'SqueezeTracy.mat')

cr2 =
    1.2365
```

像1.7.4节中的 **Save Workspace As** 和 **Save Selection As** 菜单命令一样，函数 save 会为创建的文件附加一个 .mat 扩展名。在这种情况下，产生的文件 SqueezeTracy.mat 称为 MAT 文件。MAT 文件是二进制数据文件，它包含有工作空间变量名和值。这里，它包含有一个单独的工作空间变量 c。最后，我们注意到先前计算出的压缩率 cr1 和 cr2 间的细微差别源于 MATLAB 数据文件的开销。

8.2.3 霍夫曼解码

经霍夫曼编码的图像若不能解码来创建原图像，则这种编码将毫无用处。对于前一节中的输出 y = mat2huff(x)，解码器必须首先计算用来对 x 进行编码的霍夫曼码（基于其直方图和 y 中的相关信息），然后反向映射已编码的数据（还是从 y 中提取）来重建 x。正如在下面列出的函数 x = huff2mat(y) 中看到的那样，该处理可以分为5个基本步骤：

1. 从输入结构 y 中提取维数 m 和 n 以及（最终输出 x 的）最小值 xmin。
2. 通过将 x 的直方图传递给函数 huffman，重新创建用于对 x 进行编码的霍夫曼码。重新创建的霍夫曼码在程序清单中称为 map。

```
        left = [left len + 1 len + 2];      % Add 2 unprocessed nodes
    end
end
x = unravel(y.code', link, m * n);          % Decode using C 'unravel'
x = x + xmin - 1;                           % X minimum offset adjust
x = reshape(x, m, n);                       % Make vector an array
```

如前面指出的那样，基于huff2mat的解码建立在一系列二分查找或双结果解码决策上。被扫描的经霍夫曼编码的串的每一个元素（必须是'0'或'1'），将基于转换和输出表link来触发一个二进制解码决策。link的结构使用语句link=[2;0;0]来初始化。开始三状态link数组中的每一个元素，都对应于相应单元数组code中的一个经霍夫曼编码的二进制串；起初，code = cellstr(char('','0','1'))。空串code(1)是所有的霍夫曼串解码的开始点(或初始解码状态)。link(1)中相关的2用于识别这两个可能解码的状态，这通过将'0'和'1'添加到空串中来实现。若接下来遇到的经霍夫曼编码的比特是'0'，则下一个解码状态是link(2)[因为code(2) = '0'，从而空串与'0'级联]；若是'1'，则新状态应为link(3)[在索引（2+1）或3处，code(3) ='1']。注意，相应数组link的元素为0——表明它们还未经处理，因而无法反映霍夫曼码map的正确决策。在link构造期间，若在map中找到了任何串（如'0'或'1'），即它是一个有效的霍夫曼码字，则link中相应的0会被相应的map索引中的负值（一个已解码的值）代替。否则，会插入一个新的(正值)link索引，以指向两个逻辑上(不是'00'和'01'，就是'10'和'11')遵循的新状态(可能的霍夫曼码字)。这些未经处理的新link元素会扩展link的尺寸(单元数组code也必须更新)，且结构处理会继续进行，直到link中没有未处理的元素为止。然而，huff2mat并不连续地扫描link中的未处理元素，而是维护一个称为left的跟踪数组，该数组初始化为[2,3]，随后会更新，以包含link中未经检查的元素的索引。

表8.3显示了link表，该表是为例8.2中的霍夫曼码生成的。若每个link索引都可看成是一个解码状态i，则每一个二进制编码决策（从左到右扫描已编码的串）和/或霍夫曼解码输出都由link(i)确定：

1. 若link(i)<0（如负值），则一个霍夫曼码字已被解码。解码输出是|link(i)|，其中||表示绝对值。
2. 若link(i)>0（如正值）且下一个将被处理的已编码比特是0，则下一个解码状态是索引link(i)。换言之，我们令i = link(i)。
3. 若link(i)>0且下一个将被处理的已编码比特是1，则下一个解码状态是索引link(i)+1。换言之，我们令i = link(1)+1。

表8.3 图8.3中源约简单元数组的解码表

索引 i	link(i)的值
1	2
2	4
3	−2
4	−4
5	6
6	−3
7	−1

如同前面注释的那样，正的link元素对应于二进制解码转换，而负的元素则决定解码的输出值。每当解码一个霍夫曼码字时，会在link索引i = 1处开始一个新的二分查找。对于例8.2中的已编码串101010110101…，得到的状态转换序列为i = 1,3,1,2,5,6,1,...；相应的输出序列

是-,|-2|,-,-,-,|-3|,-...，其中-号表示无输出。解码的输出值2和3是例8.2中测试图像 f2 的首列的前两个像素。

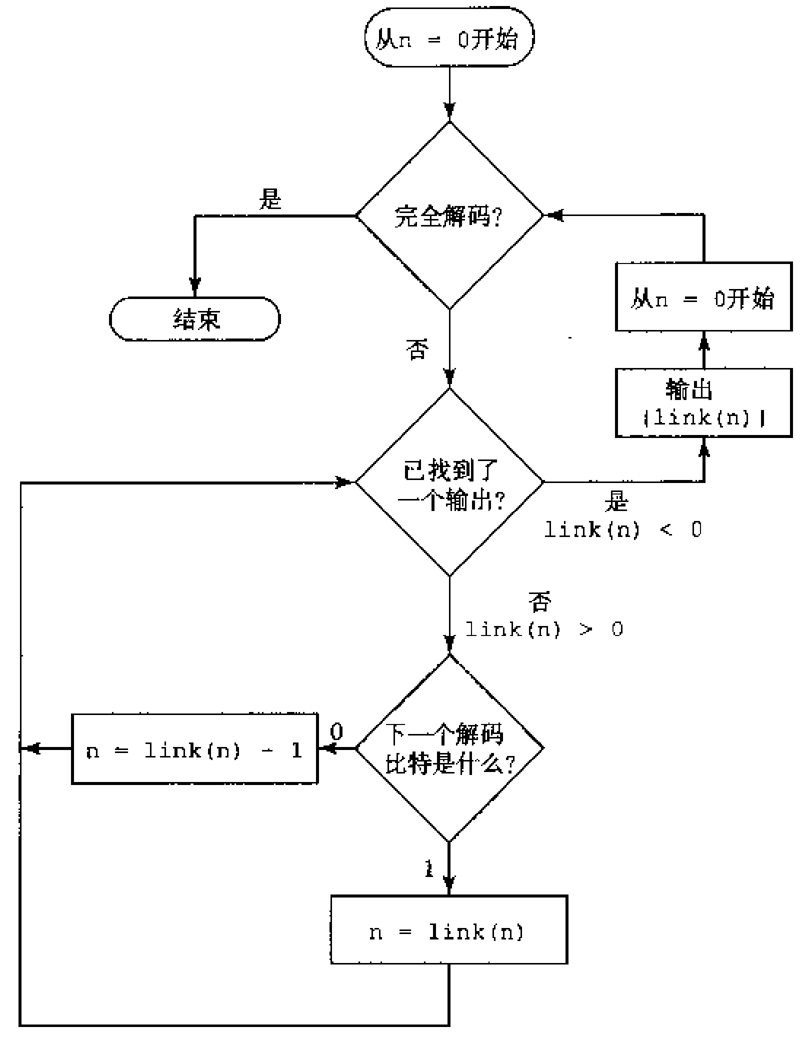


图 8.5　C 函数 unravel 的流程图

C函数unravel接受刚才描述的link结构，并使用它来驱动解码输入hx所要求的二分查找。图 8.5 以图表的形式显示了它的基本操作，随后是描述过的与表 8.3 相匹配的判决过程。但要注意的是，我们需要进行修改来补偿 C 数组是从 0 而不是 1 开始索引的这一事实。

在 MATLAN 中可结合使用 C 和 Fortran 函数，两个主要目的是：(1)在 MATLAB 中可调用已预先存在的较大的 C 和 Fortran 程序，而不必将这些程序重写成 M 文件；(2)尽管 C 和 Fortran 函数的计算速度不如 MATLAB 的 M 文件快，但以 C 和 Fortran 编码可提高效率。不管采用 C 还是 Fortran，结果函数都称为 MEX 文件；就好像它们是 M 文件或普通的 MATLAB 函数一样。然而，与 M 文件不同的是，在调用这些程序之前，必须使用 MATLAB 的 mex 脚本对它们进行编译和链接。例如，为了从 MATLAB 命令行提示符处编译和链接 Windows 平台上的函数 unravel，我们可以键入

```
>> mex unravel.c
```

这将创建名为unravel.dll且扩展名为.dll的MEX文件。若需要，提供的帮助文本必须以相同的名称保存为单独的 M 文件（其扩展名为.m)。

扩展名为.c 的 C 的 MEX 文件 unravel 的源代码如下所示：

```
/*===================================================================
 * unravel.c
 * Decodes a variable length coded bit sequence (a vector of
 * 16-bit integers) using a binary sort from the MSB to the LSB
 * (across word boundaries) based on a transition table.
 *==================================================================*/
include "mex.h"

void unravel(unsigned short *hx, double *link, double *x,
    double xsz, int hxsz)
{
  int i = 15, j = 0, k = 0, n = 0;      /* Start at root node, 1st */
                                        /* hx bit and x element */
  while (xsz - k)   {                         /* Do until x is filled */
   if (*(link + n) > 0)   {                   /* Is there a link? */
      if ((*(hx + j) >> i) & 0x0001)          /* Is bit a 1? */
          n = *(link + n);                    /* Yes, get new node */
      else n = *(link + n) - 1;               /* It's 0 so get new node */
      if (i) i--; else {j++; i = 15;}         /* Set i, j to next bit */
      if (j > hxsz)                           /* Bits left to decode? */
         mexErrMsgTxt("Out of code bits ???");
      }
else    {                               /* It must be a leaf node */
   *(x + k++) = - *(link + n);          /* Output value */
    n = 0;    }                         /* Start over at root */
}
if (k == xsz - 1)                       /* Is one left over? */
  *(x + k++) = - *(link + n);
}

void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
   double *link, *x, xsz;
   unsigned short *hx;
   int hxsz;

   /* Check inputs for reasonableness */
   if (nrhs != 3)
      mexErrMsgTxt("Three inputs required.");
   else if (nlhs > 1)
      mexErrMsgTxt("Too many output arguments.");

   /* Is last input argument a scalar? */
   if(!mxIsDouble(prhs[2])  || mxIsComplex(prhs[2])  ||
         mxGetN(prhs[2]) * mxGetM(prhs[2]) != 1)
      mexErrMsgTxt("Input XSIZE must be a scalar.");

   /* Create input matrix pointers and get scalar */
   hx = mxGetPr(prhs[0]);                /* UINT16 */
   link = mxGetPr(prhs[1]);              /* DOUBLE */
   xsz = mxGetScalar(prhs[2]);           /* DOUBLE */

   /* Get the number of elements in hx */
```

```
    hxsz = mxGetM(prhs[0]);

    /* Create 'xsz' x 1 output matrix */
    plhs[0] = mxCreateDoubleMatrix(xsz, 1, mxREAL);

    /* Get C pointer to a copy of the output matrix */
    x = mxGetPr(plhs[0]);

    /* Call the C subroutine */
    unravel(hx, link, x, xsz, hxsz);
}
```

配套的帮助文本在M文件unravel.m中提供：

```
%UNRAVEL Decodes a variable-length bit stream.
%   X = UNRAVEL(Y, LINK, XLEN) decodes UINT16 input vector Y based on
%   transition and output table LINK. The elements of Y are
%   considered to be a contiguous stream of encoded bits--i.e., the
%   MSB of one element follows the LSB of the previous element. Input
%   XLEN is the number code words in Y, and thus the size of output
%   vector X (class DOUBLE). Input LINK is a transition and output
%   table (that drives a series of binary searches):
%
%    1. LINK(0) is the entry point for decoding, i.e., state n = 0.
%    2. If LINK(n) < 0, the decoded output is |LINK(n)|; set n = 0.
%    3. If LINK(n) > 0, get the next encoded bit and transition to
%       state [LINK(n) - 1] if the bit is 0, else LINK(n).
```

像所有的C的MEX文件那样，C的MEX文件unravel.c由两个不同的部分组成：计算子程序和入口子程序。计算子程序也称为unravel，它包含有执行图8.5的基于link的解码处理的C代码。通常总是命名为mexFunction的入口程序会把C的计算子程序unravel接口到调用函数huff2mat的M文件。它基于如下内容使用MATLAB的标准MEX文件接口：

1. 四个标准的输入输出参数——nlhs，plhs，nrhs和prhs。这些参数分别是左手侧输出参量的数目（一个整数），左手侧输出参量的指针数组（所有的MATLAB数组），右手侧输出参量的数目（另一个整数），右手侧输出参量的指针数组（也是MATLAB数组）。
2. MATLAB提供了一组应用程序接口（API）函数。前缀为mx的API函数用来建立、访问、操作和/或毁坏类mxArray类的结构。例如，
   - mxCalloc可以像标准的C函数calloc一样动态地分配存储空间。相关的函数包括mxMalloc和mxRealloc，用来代替C的函数malloc和realloc。
   - mxGetScalar从输入数组prhs中提取一个标量。其他的mxGet...函数，如mxGetM，mxGetN和mxGetString，可用于提取其他类型的数据。
   - mxCreateDoubleMatrix为plhs创建一个MATLAB输出数组，其他的mxCreate...函数，如mxCreateString和mxCreateNumericArray，可用于创建其他的数据类型。

   前缀为mex的API函数执行MATLAB环境下的操作。例如，mexErrMsgTxt会将一条消息输出到MATLAB工作空间。

前述列表中的第2项中提到的API mex和mx程序的函数原型，分别保存在MATLAB头文件mex.h和matrix.h中。两者都位于<matlab>/extern/include目录中，其中<matlab>表示MATLAB在系统中的最顶级目录。头文件mex.h必须包含在所有MEX文件的开头（注意C文件的包含语句#include "mex.h"在MEX文件unravel的开头），而头文件matrix.h包含在

头文件mex.h中。包含在这些文件中的mex和mx接口程序的原型定义了那些在普通操作中使用和提供有价值内容的参数。其他信息可在MATLAB的外部接口参考手册中找到。

图8.6总结了前面的讨论，详细叙述了C的MEX文件unravel的整体结构，并且描述了它和M文件huff2mat之间的信息流。虽然概念构建在霍夫曼解码的内容中，但它们可很容易地扩展到其他基于C和/或Fortran的MATLAB函数中。

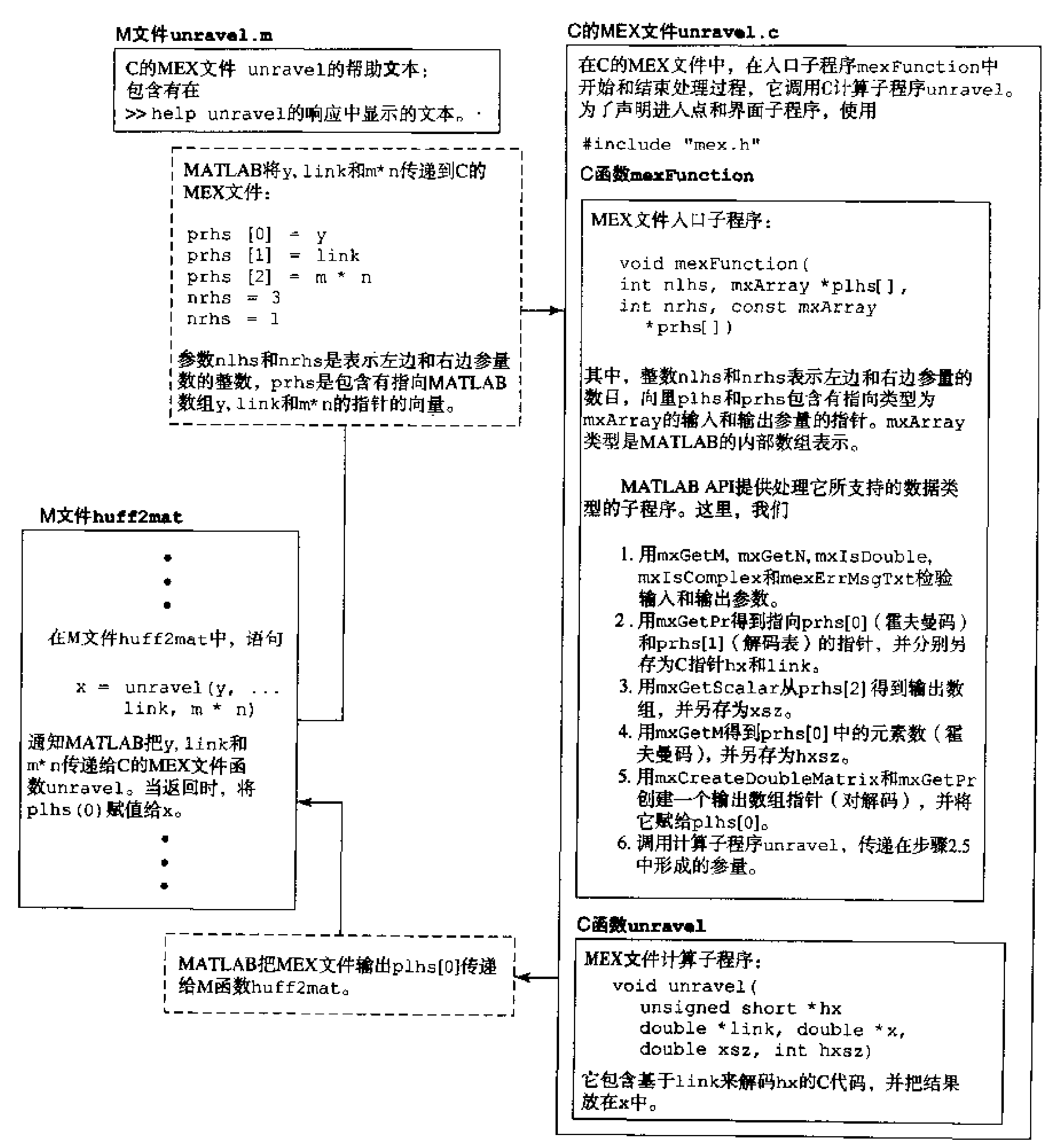


图8.6　M文件huff2mat和MATLAB可调用的C函数unravel间的交互。注意，C的MEX文件unravel包含两个函数：入口子程序mexFunction和计算子程序unravel。M文件unravel的帮助文本包含在一个单独的M文件中，该M文件的名称也为unravel

例 8.4 使用函数 huff2mat 解码

例 8.3 中经霍夫曼编码的图像可以使用下面的命令序列来解码：

```
>> load SqueezeTracy;
>> g = huff2mat(c);
>> f = imread('Tracy.tif');
>> rmse = compare(f, g)

rmse =
     0
```

注意，整个编码－解码过程都是信息保持的；原图像和解压后的图像之间的均方根误差为零。因为大部分解码工作是在 C MEX 文件 unravel 中完成的，但函数 huff2mat 要比其编码函数 mat2huff 的运行速度稍微快一些。注意，上述语句中使用了 load 函数来从例 8.3 中检索经 MAT 文件编码的输出。

8.3 像素间的冗余

考虑图 8.7(a)和图 8.7(c)所示的图像。如图 8.7(b)和图 8.7(d)所示的那样，它们实际上有着相同的直方图。我们还注意到，直方图有三种型态，这表明存在灰度级值的三个主要范围。因为图像的灰度级不是等可能出现的，所以变长编码可用于减少编码冗余，这可以通过像素的常用二进制编码来获得：

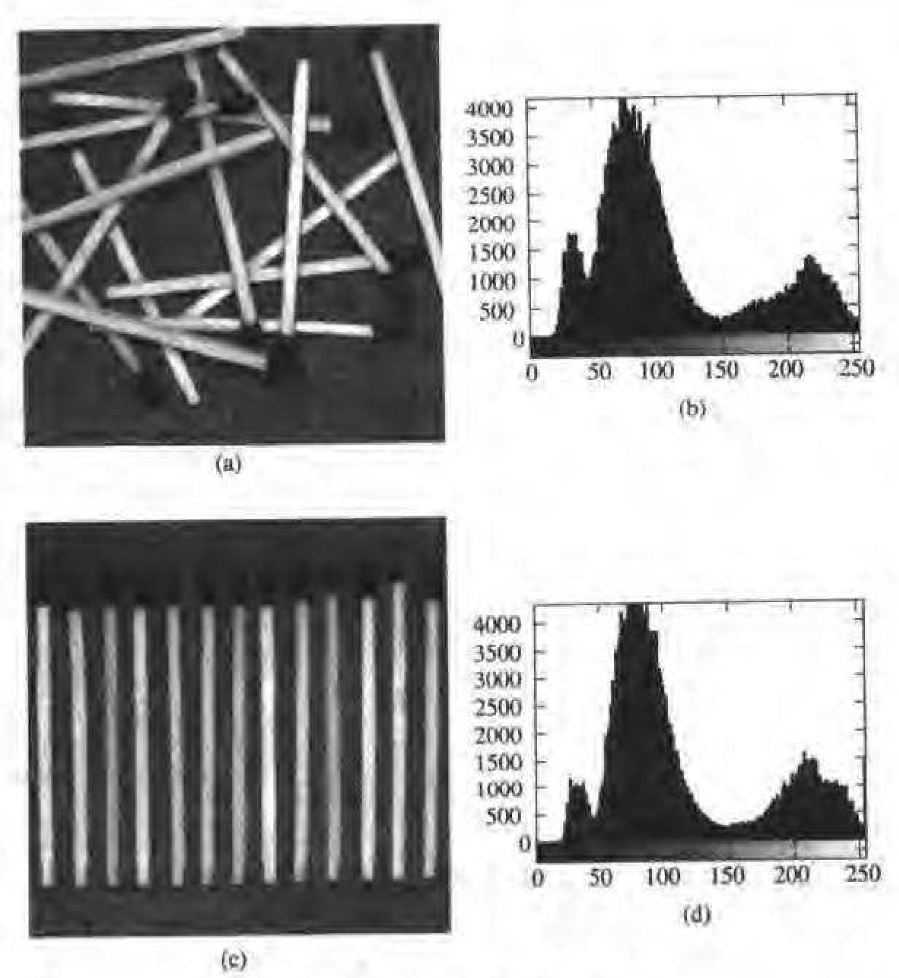


图 8.7 两幅图像及其灰度级直方图

$$e_n = f_n - \hat{f}_n$$

该误差值使用变长编码（通过符号编码器）编码，以生成压缩数据流的下一个元素。图8.8(b)所示的解码器使用收到的变长码字并执行相反的操作来重构 e_n：

$$f_n = e_n + \hat{f}_n$$

各种局部的、全局的和自适应的方法都可用于生成$\hat{f}_n$。但在大多数情形下，预测是由m个以前的像素的线性组合形成的，即

$$\hat{f}_n = \text{round}\left[\sum_{i=1}^{m} \alpha_i f_{n-i}\right]$$

其中，m是线性预测器的阶，"round" 是一个用来表示取整或获得最接近的整数的操作的函数（就像MATLAB中的 round 函数），$\alpha_i, i = 1, 2, \cdots, m$ 是预测系数。对于一维线性预测编码，该方程可重写为

$$\hat{f}(x, y) = \text{round}\left[\sum_{i=1}^{m} \alpha_i f(x, y - i)\right]$$

其中，每一个带有下标的变量现在已明确地表示为空间坐标x和y的一个函数。注意，预测$\hat{f}(x, y)$是当前扫描行上以前一些像素的函数。

M 函数mat2lpc和lpc2mat执行刚才讨论过的预测编码和解码处理（减去符号编码和解码步骤）。编码函数mat2lpc使用了一个 for 循环来在输入 x 中同时建立每一个像素的预测。在每一步迭代中，开始作为x的副本的xs向右移动了一列（左侧用0填充），乘以一个适当的预测系数，并将其加到预测和p上。因为线性预测系数的数量一般都很小，所以整个过程会比较快。注意在下面的程序清单中，若未指定预测滤波器 f，则使用一个系数为 1 的单个元素滤波器。

```
function y = mat2lpc(x, f)
%MAT2LPC Compresses a matrix using 1-D lossles predictive coding.
%   Y = MAT2LPC(X, F) encodes matrix X using 1-D lossless predictive
%   coding. A linear prediction of X is made based on the
%   coefficients in F. If F is omitted, F = 1 (for previous pixel
%   coding) is assumed. The prediction error is then computed and
%   output as encoded matrix Y.
%
%   See also LPC2MAT.

error(nargchk(1, 2, nargin));        % Check input arguments
if nargin < 2                        % Set default filter if omitted
   f = 1;
end

x = double(x);                       % Ensure double for computations
[m, n] = size(x);                    % Get dimensions of input matrix
p = zeros(m, n);                     % Init linear prediction to 0
xs = x;       zc = zeros(m, 1);      % Prepare for input shift and pad

for j = 1:length(f)                  % For each filter coefficient ...
   xs = [zc xs(:, 1:end - 1)];       % Shift and zero pad x
   p = p + f(j) * xs;                % Form partial prediotion sums
end

y = x - round (p);                   % Compute the prediction error
```

解码函数 lpc2mat 执行编码函数 mat2lpc 的反操作。在下面的程序清单中可以看到，该函数采用了一个 n 次迭代的 for 循环。其中，n 是已编码输入矩阵 y 的列数。每一次迭代都只计算已解码输出 x 中的一列，因为已解码的每一列对于后面所有列的计算都是必需的。为了减少花费在循环上的时间，在开始 for 循环之前要把 x 预分配为最大的尺寸。我们还注意到，生成预测所用的计算以函数 lpc2mat 中相同的顺序执行，以避免浮点数四舍五入误差。

```
function x = lpc2mat(y, f)
%LPC2MAT Decompresses a 1-D lossless predictive encoded matrix.
%   X = LPC2MAT(Y, F) decodes input matrix Y based on linear
%   prediction coefficients in F and the assumption of 1-D lossless
%   predictive coding. If F is omitted, filter F = 1 (for previous
%   pixel coding) is assumed.
%
%   See also MAT2LPC.

error (nargchk(1, 2, nargin));    % Check input arguments
if nargin < 2                      % Set default filter if omitted
   f = 1;
end

f = f(end:-1:1);                   % Reverse the filter coefficients
[m, n] = size(y);                  % Get dimensions of output matrix
order = length(f);                 % Get order of linear predictor
f = repmat(f, m, 1);               % Duplicate filter for vectorizing
x = zeros(m, n + order);           % Pad for 1st 'order' column decodes

% Decode the output one column at a time. Compute a prediction based
% on the 'order' previous elements and add it to the prediction
% error. The result is appended to the output matrix being built.
for j = 1:n
   jj = j + order;
   x(:, jj) = y(:, j) + round(sum(f(:, order:-1:1) .* ...
                          x(:, (jj - 1): -1:(jj - order)), 2));
end
x = x(:, order + 1:end);           % Remove left padding
```

例 8.5 无损预测编码

考虑使用简单的一阶线性预测器

$$\hat{f}(x, y) = \text{round}[\alpha f(x, y-1)]$$

对图 8.7(c)所示的图像进行编码。这种形式的预测器通常称为前像素预测器，相应的预测编码过程称为微分编码或前像素编码。图 8.9(a)显示了 $\alpha = 1$ 时产生的预测误差图像。这里，灰度级 128 对应于预测误差 0，而非零的正误差和负误差（欠估计和过估计）由函数 mat2gray 标度，以分别形成较亮的灰度或较暗的灰度：

```
>> f = imread('Aligned Matches.tif');
>> e = mat2lpc(f);
>> imshow(mat2gray(e));
>> entropy(e)

ans =
    5.9727
```

注意，预测误差的熵 e 明显地要比原图像的熵 f 小。尽管对 m 比特图像来说需要 $(m+1)$ 比特来准确地表示结果误差序列，但熵已从 7.3505 比特/像素减少到了 5.9727 比特/像素。熵的减少意味着预测误差图像可以比原图像获得更有效的编码，当然这也是映射的目的。所以，我们得到

```
>> c = mat2huff(e);
>> cr = imratio(f, c)
cr =
    1.3311
```

如同期望的那样，压缩率由 1.0821（当霍夫曼方法对灰度级直接编码时）增大到了 1.3311。预测误差 e 的直方图如图 8.9(b)所示，其计算如下：

```
>> [h, x] = hist(e(:) * 512, 512);
>> figure; bar(x, h, 'k');
```

注意，0附近出现了很大的峰值，且与输入图像的灰度级分布相比，它有着相对较小的变化[见图 8.7(d)]。如同对待前面计算的熵值那样，通过预测和差值处理消除了像素间的大量冗余。通过展示这种预测编码方案的无损特性，即通过解码 c 并将其与开始的图像 f 进行比较，我们来总结这个示例：

```
>> g = lpc2mat(huff2mat(c));
>> compare(f, g)
ans =
    0
```

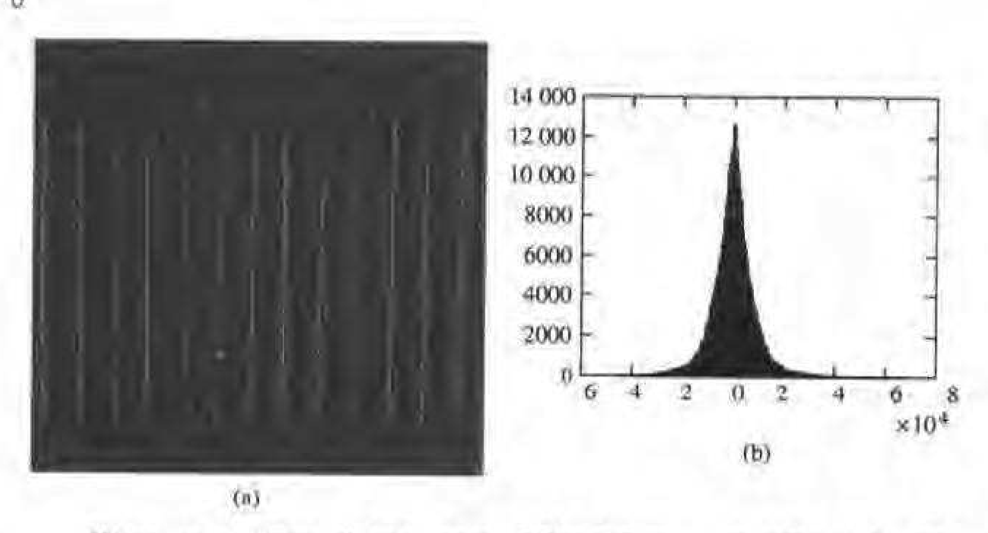


图 8.9 (a)对图 8.7(c)使用 f = [1]的预测误差图像；(b)预测误差的直方图

8.4 心理视觉冗余

与编码及像素间冗余不同，心理视觉冗余与真实的或可计量的视觉信息有关。消除这种冗余是值得的，因为对于通常的视觉处理来说，信息本身不完全是必需的。因为心理视觉冗余数据的消除会导致量化信息的损失，所以将其称为量化。该术语通常表示把宽范围的输入值映射为有限数量的输出值。因为量化是一个不可逆的操作（如视觉信息损失），所以它导致了数据的有损压缩。

例8.6 通过量化压缩

考虑图8.10中的图像。图8.10(a)显示了一幅有着256级灰度的单色图像。图8.10(b)是均匀量化为4比特或16个可能灰度级的同一幅图像。导致的压缩率为2:1。注意，伪轮廓已在原图像先前平滑的区域出现。这是更为粗糙地表示该图像的灰度级是导致的正常视觉效果。

图8.10(c)示例了在使用量化并利用人类视觉系统的特性后所出现的重大改进。虽然第二种量化的压缩率仍是2:1，但是伪轮廓减少了很多，取而代之的是一些不则明显的粒状物。注意,无论是哪一种情况，解压缩都是不必要、而且是不可能的（即量化是一种不可逆的操作）。

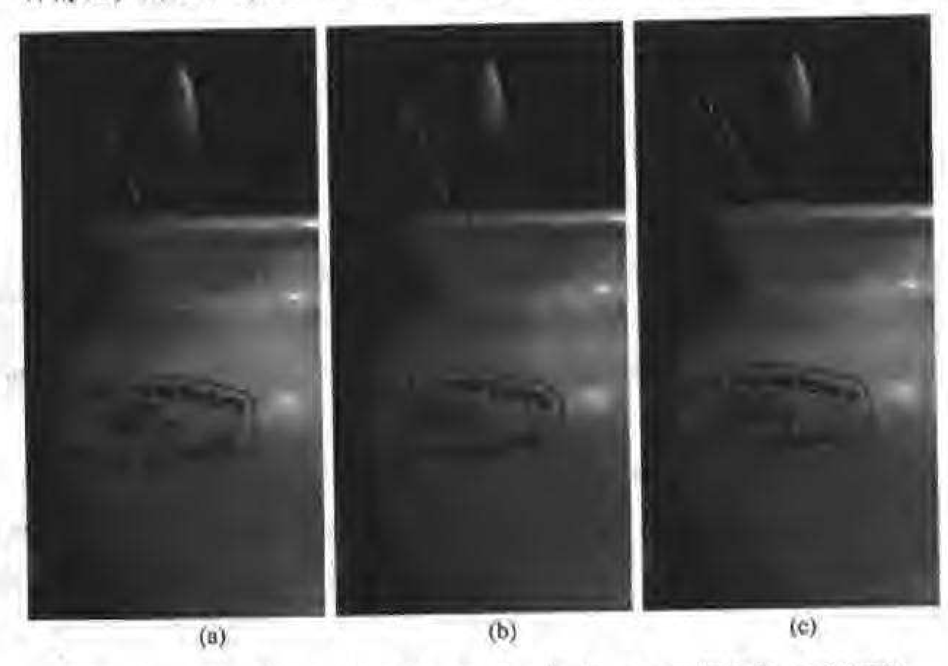


图8.10 (a)原图像；(b)均匀量化为16级灰度；(c)IGS量化为16级灰度

用于产生图8.10(c)所用的方法称为改进的灰度级（IGS）量化。该方法可识别人眼对边缘的固有敏感性，并通过对每一个像素增加一个伪随机数来消除它，伪随机数在量化之前由相邻像素的低阶比特产生。因为低阶比特是完全随机的，所以这相当于在正常情况下对与伪轮廓有关的人为边缘加入一个随机性的灰度级（它依赖于图像的局部特征）。下面列出的量化函数quantize用于执行IGS量化和传统的低阶比特截断。注意，IGS实现是向量化的，所以输入x一次只处理一列。为了生成图8.10(c)中的一列4比特，可根据x的一列的和与现有的（先前产生的）4个最低有效位的和形成一列总和s（最初全部设为0）。若任何x值中的4个最高有效位是1111_2，则总是加上0000_2。然后，所得的和中的4个最高有效位将用做正被处理列的编码像素值。

```
function y = quantize(x, b, type)
%QUANTIZE Quantizes the elements of an UINT8 matrix.
%   Y = QUANTIZE(X, B, TYPE) quantizes X to B bits. Truncation is
%   used unless TYPE is 'igs' for Improved Gray Scale quantization.

error(nargchk(2, 3, nargin));        % Check input arguments
if ndims(x) ~= 2 | ~isreal(x) | ...
   ~isnumeric(x) | ~isa(x, 'uint8')
   error('The input must be a UINT8 numeric matrix.');
end

% Create bit masks for the quantization
lo = uint8 (2 ^ (8 - b) - 1);
```

8.5 JPEG压缩

前面一节中介绍的技术直接操作一幅图像的像素，因而是空间域方法。在这一节中，我们将讨论一类流行的压缩标准，它们是以修改图像的变换为基础的。我们的目的是在图像压缩中引入二维变换，提供更多关于减少图像冗余的例子，并介绍图像压缩技术的进展情况。已有的标准（尽管我们只考虑它们的近似）设计用于处理较宽范围的图像类型和压缩需求。

在变换编码中，可逆的线性变换可以是第4章中的DFT或离散余弦变换（DCT）

$$T(u,v)=\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)\alpha(u)\alpha(v)\cos\left[\frac{(2x+1)u\pi}{2M}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

其中，

$$\alpha(u)=\begin{cases}\sqrt{\dfrac{1}{M}} & u=0\\ \sqrt{\dfrac{2}{M}} & u=1,2,\cdots,M-1\end{cases}$$

[$\alpha(v)$与此类似]用于把一幅图像映射成为一组变换系数，然后对系数进行量化和编码。对于大多数的正常图像来说，多数系数具有较小的数值且可被粗略地量化（或者完全抛弃），而产生的图像失真较小。

8.5.1 JPEG

最流行且最全面的连续色调静止画面压缩标准之一是JPEG(Joint Photographic Experts Group，联合图像专家组）标准。在JPEG基准编码系统（该系统基于离散余弦变换，且对于大多数压缩应用来说是足够的）中，输入和输出图像都限制为8比特图像，而量化的DCT系数值限制在11比特。在图8.11(a)所示的简化方框图中可以看到，压缩本身分4步执行：8 × 8子图像提取、DCT计算、量化以及变长码分配。

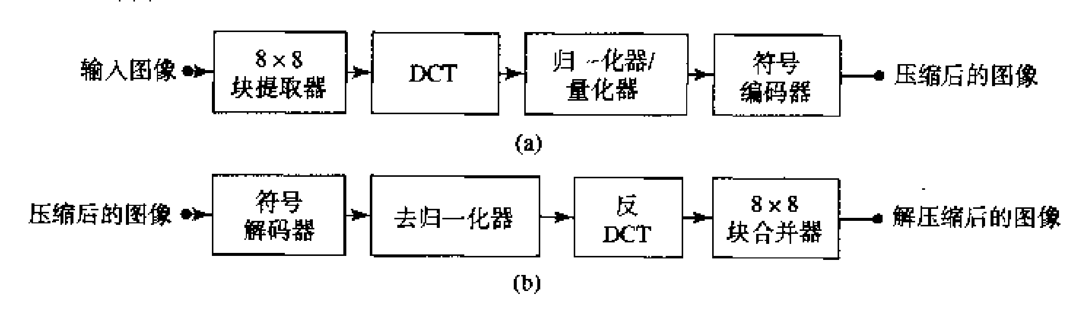


图8.11 JPEG框图：(a) 编码器；(b) 解码器

JPEG压缩处理的第一步是把输入图像细分为不重叠的8 × 8像素块。随后从左到右、从上到下进行处理。若每一个8 × 8块或子图像都得到了处理，则其64个像素将通过减去2^{m-1}来做灰度级移动（其中2^m是图像中的灰度级数），并且计算其二维离散余弦变换。得到的系数然后根据下式被归一化和量化，

$$\hat{T}(u,v)=\text{round}\left[\frac{T(u,v)}{Z(u,v)}\right]$$

其中，$\hat{T}(u,v)$ $(u,v=0,1,\cdots,7)$是导致的归一化和量化系数，$T(u,v)$是图像$f(x,y)$的一个8 × 8块的DCT，$Z(u,v)$是一个类似于图8.12(a)的变换归一化数组。通过缩放$Z(u,v)$，可以得到各种压缩率，且重建的图像的质量可以得到保证。

量化完每一块的DCT系数后，可使用图8.12(b)所示的zigzag模式来重新排列$\hat{T}(u, v)$的元素。因为得到的（量化系数的）一维重排数组是根据渐增的空间频率来排列的，所以图8.11(a)中的符号编码器可充分利用重新排列所导致的零的长游程。特别地，非零AC系数［即所有的$\hat{T}(u, v)$，但$u = v = 0$除外］用一个变长码来编码，该变长码定义了系数的值和前面0的个数。DC系数［即$\hat{T}(0, 0)$］是相对于前一个子图像的DC系数的编码差。默认的AC和DC霍夫曼编码表可由该标准提供，但用户可以免费构建自定义表和标准化数组，它们实际上可能适合于将被压缩图像的特征。

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

图8.12 (a)默认的JPEG标准化数组；(b)JPEG的zigzag系数排列顺序

尽管JPEG标准的完整实现已超出了本章讨论的范围，但下面的M文件近似了基准编码处理：

```
function y = im2jpeg(x, quality)
%IM2JPEG Compresses an image using a JPEG approximation.
%   Y = IM2JPEG(X, QUALITY) compresses image X based on 8 x 8 DCT
%   transforms, coefficient quantization, and Huffman symbol
%   coding. Input QUALITY determines the amount of information that
%   is lost and compression achieved. Y is an encoding structure
%   containing fields:
%
%     Y.size        Size of X
%     Y.numblocks   Number of 8-by-8 encoded blocks
%     Y.quality     Quality factor (as percent)
%     Y.huffman     Huffman encoding structure, as returned by
%                   MAT2HUFF
%
% See also JPEG2IM.

error(nargchk(1, 2, nargin));              % Check input arguments
if ndims(x) ~= 2 | ~isreal(x) | ~isnumeric(x) | ~isa(x, 'uint8')
   error ('The input must be a UINT8 image.');
end
if nargin < 2
   quality = 1;    % Default value for quality.
end

m = [16 11 10 16 24 40 51 61       % JPEG normalizing array
     12 12 14 19 26 58 60 55       % and zig-zag redordering
     14 13 16 24 40 57 69 56       % pattern.
     14 17 22 29 51 87 80 62
     18 22 37 56 68 109 103 77
```

```
      24 35 55 64 81 104 113 92
      49 64 78 87 103 121 120 101
      72 92 95 98 112 100 103 99] * quality;

order = [1 9  2  3  10 17 25 18 11 4  5  12 19 26 33  ...
        41 34 27 20 13 6  7  14 21 28 35 42 49 57 50  ...
        43 36 29 22 15 8  16 23 30 37 44 51 58 59 52  ...
        45 38 31 24 32 39 46 53 60 61 54 47 40 48 55  ...
        62 63 56 64];

[xm, xn] = size(x);                    % Get input size.
x = double(x) - 128;                   % Level shift input
t = dctmtx(8);                         % Compute 8 x 8 DCT matrix

% Compute DCTs of 8x8 blocks and quantize the coefficients.
y = blkproc(x, [8 8], 'P1 * x * P2', t, t');
y = blkproc(y, [8 8], 'round(x ./ P1)', m);

y = im2col(y, [8 8], 'distinct');    % Break 8x8 blocks into columns
xb = size(y, 2);                       % Get number of blocks
y = y(order, :);                       % Reorder column elements

eob = max(x(:)) + 1;                   % Create end-of-block symbol
r = zeros(numel(y) + size(y,2), 1);
count = 0;
for j = 1:xb                           % Process 1 block (col) at a time
   i = max(find(y(:,j)));              % Find last non-zero element
   if isempty(i)                       % No nonzero block values
      i = 0;
   end
   p = count + 1;
   q = p + i;
   r(p:q) = [y(1:i, j); eob];          % Truncate trailing 0's, add EOB,
   count = count + i + 1;              % and add to output vector
end
r((count + 1):end) = [];               % Delete unusued portion of r

y.size      = uint16([xm xn]);
y.numblocks = uint16(xb);
y.quality   = uint16(quality * 100);
y.huffman   = mat2huff(r);
```

为与图8.11(a)所示的框图一致，对于输入图像x的不同8×8部分或块，函数im2jpeg一次处理一块（而不是一次处理整个图像）。两个专门的块处理函数（blkproc和im2col）可用于简化该计算。函数blkproc的标准语法为

```
B = blkproc(A, [M N], FUN, P1, P2, ...),
```

该函数自动实现图像块处理的整个过程。函数blkproc的参量为：一幅输入图像A，将被处理的块的大小[M N]，用于处理这些块的函数（FUN），以及块处理函数FUN的一些可选输入参数P1,P2,...。函数blkproc然后把A分成$M\times N$个块（需要时填充零），对每个块调用参数为P1,P2,...的函数FUN，并重新将结果组合到输出图像B。

第二个用于im2jpeg的块处理函数是im2col。当blkproc不适合执行专门的面向块的操作时，im2col经常用来重新排列输入，以便操作可以以一种更为简单且高效的方式来编码（例如，通过允许操作向量化）。函数im2col的输出是一个矩阵，其中每一列都包含输入图像的一个特定块的元素。它的标准格式为

```
for k = 1:length(order)
   rev(k) = find(order == k);
end

m = double(y.quality) / 100 * m;          % Get endoding quality.
xb = double(y.numblocks);                 % Get x blocks.
sz = double(y.size);
xn = sz(2);                               % Get x columns.
xm = sz(1);                               % Get x rows.
x = huff2mat(y.huffman);                  % Huffman decode.
eob = max(x(:));                          % Get end-of-block symbol

z = zeros(64, xb);   k = 1;               % Form block columns by copying
for j = 1:xb                              % successive values from x into
   for i = 1:64                           % columns of z, while changing
      if x(k) == eob                      % to the next column whenever
         k = k + 1; break;                % an EOB symbol is found.
      else
         z(i, j) = x(k);
         k = k + 1;
      end
   end
end

z = z(rev, :);                                       % Restore order
x = col2im(z, [8 8], [xm xn], 'distinct');           % Form matrix blocks
x = blkproc(x, [8 8], 'x .* P1', m);                 % Denormalize DCT
t = dctmtx(8);                                       % Get 8 x 8 DCT matrix
x = blkproc(x, [8 8], 'P1 * x * P2', t', t);         % Compute block DCT-1
x = uint8(x + 128);                                  % Level shift
```

从矩阵 z 的列重建一幅二维图像矩阵 z，其中每个64元素列都是重构图像中的一个8 × 8块。参数 A，B，[M N]和 'distinct' 的定义与它们在函数 im2col 中的定义相同，而数组[MM NN]表示输出图像 A 的维数。

例 8.8 JPEG 压缩

图 8.13(a)和图 8.13(b)分别显示了图 8.4(a)所示单色图像的 JPEG 编码图像和解码图像。压缩率为 18 : 1 的第一幅图像是直接应用图 8.12(a)中的归一化矩阵得到的；压缩率为 42 : 1 的第二幅图像是由标准化数组乘以 4（缩放）产生的。

图 8.4(a)所示原图像与图 8.13(a)和图 8.13(b)所示重构图像之间的差别分别显示在图 8.13(c)和图 8.13(d)中。每幅图像都被放大，以便使误差更为明显。相应的均方根误差是 2.5 和 4.4 灰度级。这些误差对图像质量所造成的影响在图 8.13(e)和图 8.13(f)中表现得更为明显。这些图像分别显示了图 8.13(a)和图 8.13(b)的放大部分，并且可以更好地评估重构图像间的细微差别。图 8.4(b)显示了放大后的原图像。注意，在两幅放大后的近似图像中，出现了分块效应。

图 8.13 中的图像和刚讨论过的数值结果是由下列命令产生的：

```
>> f = imread('Tracy.tif');
>> c1 = im2jpeg(f);
>> f1 = jpeg2im(c1);
>> imratio(f, c1)
ans =
    18.2450

>> compare(f, f1, 3)
```

```
ans =
    2.4675

>> c4 = im2jpeg(f, 4);
>> f4 = jpeg2im(c4);
>> imratio(f, c4)

ans =
    41.7826

>> compare(f, f4, 3)

ans =
    4.4184
```

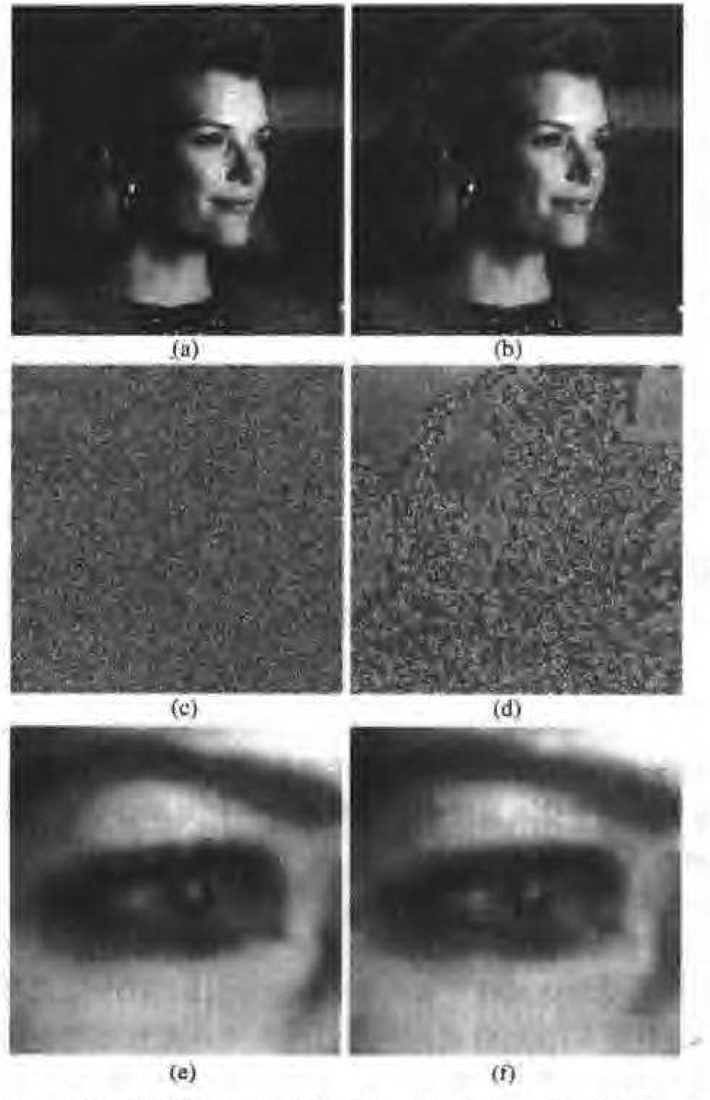


图 8.13 左列：使用图 8.12(a)所示的标准化数组和 DCT 的图 8.4 的近似图像。右列：标准化矩阵放大 4 倍后的类似结果

这些结果不同于那些在真实 JPEG 基准编码环境中获得的结果，因为 `im2jpeg` 只是近似了 JPEG 标准的霍夫曼编码处理。值得注意的两个主要差别是：(1)在该标准中，所有系数零的游程都进行了霍夫曼编码，而 `im2jpeg` 只对每个块的最后游程编码；(2)标准的编码器和解码器基于已知的（默认的）霍夫曼码，而 `im2jpeg` 携带有用于重构图像上编码霍夫曼码字的信息。利用这个标准，上面给出的压缩率几乎可以加倍。

8.5.2 JPEG 2000

像前一节的初始JPEG版本一样，JPEG 2000基于这样的概念，即解除了图像中像素相关性的变换系数可以比原始像素本身更有效地编码。若变换的基本函数在JPEG 2000的情况下是小波，则可以把大部分重要的视觉信息集中在少数的系数中，而剩下的系数可以粗略地进行量化，或是将其截短为零，从而只会产生很小的图像失真。

图 8.14 显示了一个简化的 JPEG 2000 编码系统（缺少了几个可选操作）。像在最初的 JPEG 标准中那样，编码处理的第一步是通过减去 2^{m-1}（2^m是图像中灰度级的数目）来进行图像的灰度级移动。然后可计算图像的行和列的一维离散小波变换。对无损压缩，所用的变换是双正交的，采用的是 5-3 系数尺度和小波向量。在有损应用中，使用的是 9-7 系数尺度的小波向量（见第 7 章中的函数 `wavefilter`）。对于任何一种情况，最初的分解产生了 4 个子带，即图像的低分辨率近似以及图像的水平、垂直和对角线频率特征。

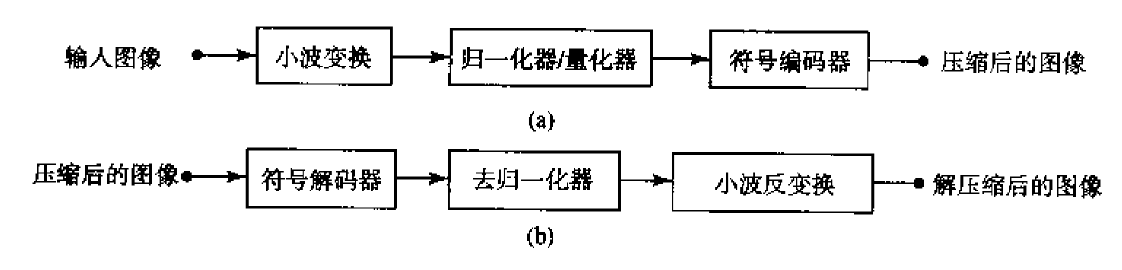


图 8.14 JPEG 2000 的方框图：(a)编码器；(b)解码器

重复分解步骤N_L次，通过将后续的迭代严格限制为先前分解的近似系数，重复分解步骤次将产生一个N_L尺度的小波变换。邻近的尺度与 2 的幂相关联，且最低的尺度只包括原图像的明确定义的近似。从图 8.15 中可以看出，$N_L = 2$ 的情形下总结出了标准的表示法，一般的N_L尺度变换包含 $3N_L + 1$ 个子带，这些子带的系数表示为 a_b，其中 $b = N_LLL$，$N_LHL, \cdots, 1HL, 1LH, 1HH$。该标准未指定将被计算的尺度的数量。

在计算N_L尺度小波变换后，变换系数的总数等于原图像中的样本数——但重要的可视集中少数系数中。为了减少表示它们所需要的比特数，子带b的系数$a_b(u, v)$可利用下述公式量化为值$q_b(u, v)$：

$$q_b(u, v) = \text{sign}[a_b(u, v)] \cdot \text{floor}\left[\frac{|a_b(u, v)|}{\Delta_b}\right]$$

其中，“sign”和“floor”操作符的作用类似于有着相同名称的 MATLAB 函数（如函数 `sign` 和 `floor`）。量化步长为

$$\Delta_b = 2^{R_b - \varepsilon_b}\left(1 + \frac{\mu_b}{2^{11}}\right)$$

其中，R_b是子带b的标称动态范围，ε_b和μ_b是分配给子带系数的指数和尾数的比特数。子带b的标称动态范围是用于表示原图像的比特数和用于子带b的分析增益比特数之和。子带分析增益比特遵循图 8.15 所示的简单模式。例如，子带 $b = 1HH$ 有两个分析增益比特。

对无损压缩来说，$\mu_b = 0$ 和 $R_b = \varepsilon_b$，所以 $\Delta_b = 1$。对于不可逆压缩来说，未指定特殊的量化步长。相反，在子带基础上必须为解码器提供指数和尾数的比特数，这称为显式量化，或对于 N_LLL 子带来说，称为隐式量化。在后一种情况下，剩下的子带使用外推的 N_LLL 子带参数来量化。若令 R_b 和 μ_b 是分配给 N_LLL 子带的比特数，则子带 b 的外推参数是

$$\mu_b = \mu_0$$
$$\varepsilon_b = \varepsilon_0 + nsd_b - nsd_0$$

其中，nsd_b表示从原图像到子带b的子带分解级数。编码处理的最后步骤是在比特平面的基础上对量化的系数进行算术编码。尽管本章未讨论算术编码，但算术编码是一个类似于霍夫曼编码的变长编码过程，用于减少编码的冗余。

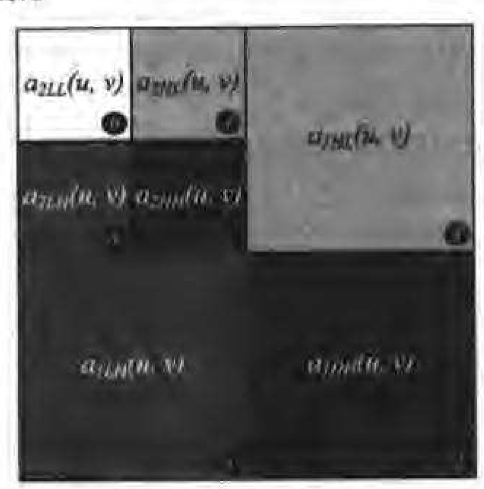


图8.15 JPEG 2000两尺度小波变换系数表示和分析增益（在圆中）

自定义函数im2jpeg2k可近似图8.14(a)的JPEG 2000编码处理，但算术符号编码除上。就像我们在下面的程序清单中所见的那样，为简单起见，已使用零游程编码来代替了霍夫曼编码。

```
function y = im2jpeg2k(x, n, q)
% IM2JPEG2K Compresses an image using a JPEG 2000 approximation.
%   Y = IM2JPEG2K(X, N, Q) compresses image X using an N-scale JPEG
%   2K wavelet transform, implicit or explicit coefficient
%   quantization, and Huffman symbol coding augmented by zero
%   run-length coding. If quantization vector Q contains two
%   elements, they are assumed to be implicit quantization
%   parameters; else, it is assumed to contain explicit subband step
%   sizes. Y is an encoding structure containing Huffman-encoded
%   data and additional parameters needed by JPEG2K2IM for decoding.
%
%   See also JPEG2K2IM.

global RUNS

error(nargchk(3, 3, nargin));          % Check input arguments

if ndims(x) ~= 2 | ~isreal(x) | ~isnumeric(x) | ~isa(x, 'uint8')
   error('The input must be a UINT8 image.');
end

if length(q) ~= 2 & length(q) ~= 3 * n + 1
   error('The quantization step size vector is bad.');
end
% Level shift the input and compute its wavelet transform
x = double(x) - 128;
[c, s] = wavefast(x, n, 'jpeg9.7');

% Quantize the wavelet coefficients.
q = stepsize(n, q);
sgn = sign(c);    sgn(find(sgn == 0)) = 1;    c = abs(c);
for k = 1:n
   qi = 3 * k - 2;
```

```
   c = wavepaste('h', c, s, k, wavecopy('h', c, s, k) / q(qi));
   c = wavepaste('v', c, s, k, wavecopy('v', c, s, k) / q(qi + 1));
   c = wavepaste('d', c, s, k, wavecopy('d', c, s, k) / q(qi + 2));
end
c = wavepaste('a', c, s, k, wavecopy('a', c, s, k) / q(qi + 3));
c = floor(c);     c = c .* sgn;

% Run-length code zero runs of more than 10. Begin by creating
% a special code for 0 runs ('zrc') and end-of-code ('eoc') and
% making a run-length table.
zrc = min(c(:)) - 1;    eoc = zrc - 1;    RUNS = [65535];

% Find the run transition points: 'plus' contains the index of the
% start of a zero run; the corresponding 'minus' is its end + 1.
z = c == 0;                z = z - [0 z(1:end - 1)];
plus = find(z == 1);       minus = find(z == -1);

% Remove any terminating zero run from 'c'.
if length(plus) ~= length(minus)
   c(plus(end):end) = [];    c = [c eoc];
end

% Remove all other zero runs (based on 'plus' and 'minus') from 'c'.
for i = length(minus):-1:1
   run = minus(i) - plus(i);
   if run > 10
      ovrflo = floor(run / 65535);    run = run - ovrflo * 65535;
      c = [c(1:plus(i) - 1) repmat([zrc 1], 1, ovrflo) zrc ...
           runcode(run) c(minus(i):end)];
   end
end

% Huffman encode and add misc. information for decoding.
y.runs    = uint16(RUNS);
y.s       = uint16(s(:));
y.zrc     = uint16(-zrc);
y.q       = uint16(100 * q');
y.n       = uint16(n);
y.huffman = mat2huff(c);

%-------------------------------------------------------------------%
function y = runcode(x)
% Find a zero run in the run-length table. If not found, create a
% new entry in the table. Return the index of the run.
global RUNS
y = find(RUNS == x);
if length(y) ~= 1
   RUNS = [RUNS; x];
   y = length(RUNS);
end

%-------------------------------------------------------------------%
function q = stepsize(n, p)
% Create a subband quantization array of step sizes ordered by
% decomposition (first to last) and subband (horizontal, vertical,
% diagonal, and for final decomposition the approximation subband).

if length(p) == 2            % Implicit Quantization
   q = [];
```

```
    qn = 2 ^ (8 - p(2) + n) * (1 + p(1) / 2 ^ 11);
    for k = 1:n
       qk = 2 ^ - k * qn;
       q = [q (2 * qk) (2 * qk) (4 * qk)];
    end
    q = [q qk];
else                              % Explicit Quantization
    q = p;
end

q = round(q * 100) / 100;         % Round to 1/100th place
if any(100 * q > 65535)
    error('The quantizing steps are not UNIT16 representable.');
end
if any(q == 0)
    error('A quantizing step of 0 is not allowed.');
end
```

JPEG 2000解码器只是简单地颠倒了前面讨论的操作。在解码经算术编码的系数后，重构了原图像的一个用户选择的子带数。虽然编码器已对特定的子带算术编码了 M_b 个比特平面，但用户可根据代码流的嵌入属性仅选择解码 N_b 个比特平面。这相当于使用大小为 $2^{M_b-N_b}\cdot\Delta_b$ 的步长来量化系数。任何未解码的比特设置为零，且得到的系数 $\overline{q}_b(u,v)$ 可使用下式来去归一化：

$$R_{q_b}(u,v)=\begin{cases}(\overline{q}_b(u,v)+2^{M_b-N_b(u,v)})\cdot\Delta_b & \overline{q}_b(u,v)>0\\(\overline{q}_b(u,v)-2^{M_b-N_b(u,v)})\cdot\Delta_b & \overline{q}_b(u,v)<0\\0 & \overline{q}_b(u,v)=0\end{cases}$$

其中，$R_{qb}(u,v)$表示一个去归一化变换系数，$N_b(u,v)$是 $\overline{q}_b(u,v)$ 的解码比特平面的数目。然后对去归一化的系数进行反变换和灰度级移动，以便产生原图像的一个近似。自定义函数jpeg2k2im可以近似这个过程，即与前面使用函数im2jpeg2k进行压缩相反的过程。

```
function x = jpeg2k2im(y)
%JPEG2K2IM Decodes an IM2JPEG2K compressed image.
%   X = JPEG2K2IM(Y) decodes compressed image Y, reconstructing an
%   approximation of the original image X.  Y is an encoding
%   structure returned by IM2JPEG2K.
%
% See also IM2JPEG2K.

error(nargchk(1, 1, nargin));        % Check input arguments

% Get decoding parameters: scale, quantization vector, run-length
% table size, zero run code, end-of-data code, wavelet bookkeeping
% array, and run-length table.
n = double(y.n);
q = double(y.q) / 100;
runs = double(y.runs);
rlen = length(runs);
zrc = -double(y.zrc);
eoc = zrc - 1;
s = double(y.s);
s = reshape(s, n + 2, 2);

% Compute the size of the wavelet transform.
cl = prod(s(1, :));
```

```
for i = 2:n + 1
   cl = cl + 3 * prod(s(i, :));
end

% Perform Huffman decoding followed by zero run decoding.
r = huff2mat(y.huffman);

c = [];    zi = find(r == zrc);    i = 1;
for j = 1:length(zi)
   c = [c r(i:zi(j) - 1) zeros(1, runs(r(zi(j) + 1)))];
   i = zi(j) + 2;
end

zi = find(r == eoc);                  % Undo terminating zero run
if length(zi) == 1                    % or last non-zero run.
   c = [c r(i:zi - 1)];
   c = [c zeros(1, cl - length(c))];
else
   c = [c r(i:end)];
end

% Denormalize the coefficients.
c = c + (c > 0) - (c < 0);
for k = 1:n
   qi = 3 * k - 2;
   c = wavepaste('h', c, s, k, wavecopy('h', c, s, k) * q(qi));
   c = wavepaste('v', c, s, k, wavecopy('v', c, s, k) * q(qi + 1));
   c = wavepaste('d', c, s, k, wavecopy('d', c, s, k) * q(qi + 2));
end
c = wavepaste('a', c, s, k, wavecopy('a', c, s, k) * q(qi + 3));
% Compute the inverse wavelet transform and level shift.
x = waveback(c, s, 'jpeg9.7', n);
x = uint8(x + 128);
```

图8.14中基于小波的JPEG 2000系统与图8.11中基于DCT的JPEG系统间的主要不同，在于后者省略了子图像处理级。因为小波变换具有计算高效和固有的局部化特点（即它们的基本函数限制在持续时间内），所以无须将图像细分为子块。就像我们在下面的例子中所看到的那样，省略细分步骤可消除在高压缩率下表征基于DCT的近似的分块效应。

例8.9　JPEG 2000压缩

图8.16显示了图8.4(a)所示单色图像的两个JPEG 2000近似。图8.16(a)是由压缩比为42 : 1的编码重构的图像；图8.16(b)是由压缩比为88 : 1的编码产生的。这两幅结果图像分别是使用5尺度变换以及$\mu_0 = 8$与$\varepsilon_0 = 8.5$和7的隐式量化获得的。因为im2jpeg2k只近似JPEG 2000的面向比特平面的算术编码，所以上面提到的压缩率不同于JPEG 2000编码器得到的真实结果。事实上，真实的比率会以因子2增加。

因为比率为42 : 1，所以图8.16左列中的压缩结果与图8.13（例8.8）右列的压缩结果相同，而图8.16(a)、图8.16(c)和图8.16(e)可以在质量和数量两方面与图8.13(b)、图8.13(d)和图8.13(f)的基于变换的JPEG结果进行比较。可以看出，基于小波的JPEG 2000图像的误差得到了明显降低。事实上，图8.16(a)所示的基于JPEG 2000的结果的均方根误差是3.7灰度级，而图8.13(b)所示的基于变换的JPEG结果的均方根误差是4.4灰度级。除了可降低重构误差外，基于JPEG 2000的编码明显地增强了图像的质量。这种质量增强在图8.16(e)中表现得更为明显。注意，在图8.13(f)所示的基于变换的结果中，明显的分块效应已不再存在。

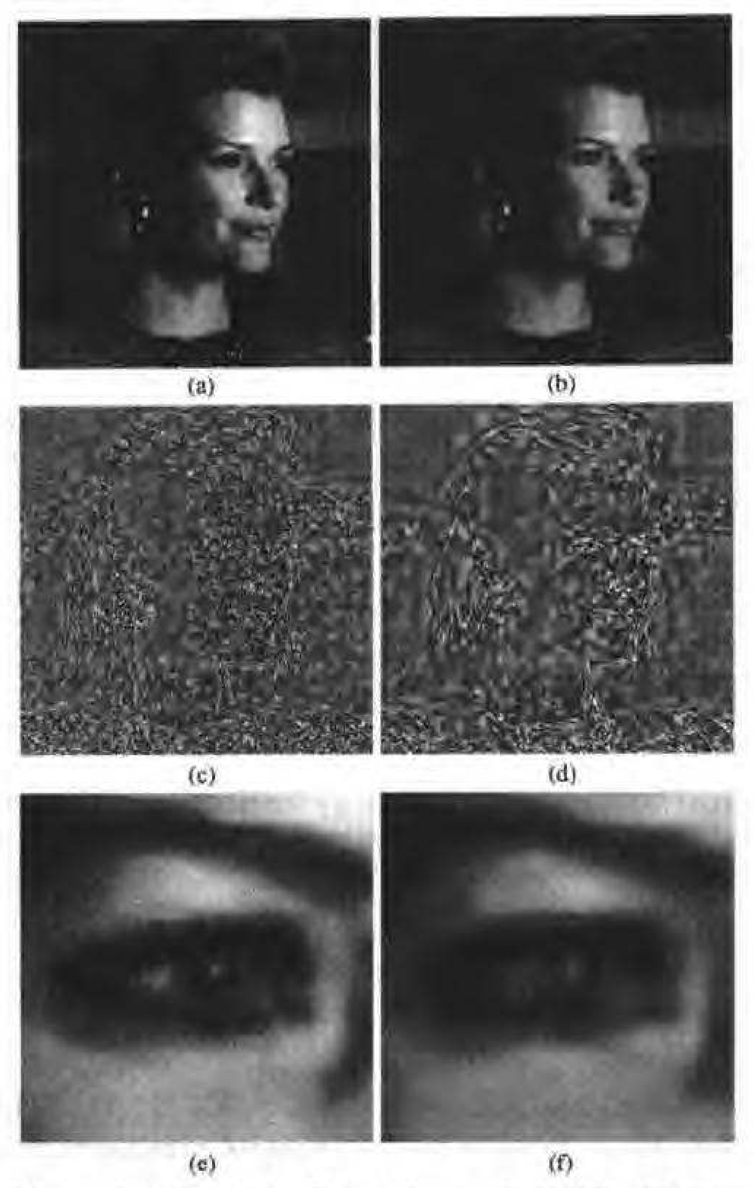


图 8.16　左列：图 8.4 的 JPEG 2000 近似，其中采用了 5 尺度变换以及 $\mu_0 = 8$ 和 $\varepsilon_0 = 8.5$ 的隐式量化；右列：$\varepsilon_0 = 7$ 时得到的类似结果

当压缩的量级增加到 88 : 1 时，如图 8.16(b)所示的那样，女士衣服的纹理出现了一些损失，而且她的眼睛变得模糊了。这两种效应可在图 8.16(b)和图 8.16(f)中见到。这些重构的均方根误差大约是 5.9 灰度级。图 8.16 所示的结果是用如下命令生成的：

```
> f = imread('Tracy.tif');
> c1 = im2jpeg2k(f, 5, [8 8.5]);
>> f1 = jpeg2k2im(c1);
>> rms1 = compare(f, f1)

rms1 =
    3.6931

>> cr1 = imratio(f, c1)
cr1 =
    42.1589

>> c2 =im2jpeg2k(f, 5, [8 7]);
```

```
>> f2 = jpeg2k2im(c2);
>> rms2 = compare(f, f2)

rms2 =
    5.9172

>> cr2 = imratio(f, c2)
cr2 =
    87.7323
```

注意，当提供一个两元素向量作为函数im2jpeg2k的第三个参量时，会使用隐式量化。若该向量的长度不是2，则该函数会使用显式量化，并且必须提供大小为$3N_L+1$的步长（其中N_L是将被计算的尺度数）。对于分解的每一个子带，步长为1；它们必须按分解级别（第一、第二、第三，等等）和子带类型（即水平、垂直、对角线和近似值）排列。例如，

```
> c3 =im2jpeg2k(f, 1, [1 1 1 1]);
```

计算一个1尺度变换，并使用显式量化——所有4个子带都使用步长$\Delta_1=1$来量化。换言之，变换系数已四舍五入为最接近的整数。这是im2jpeg2k实现的最小误差情况，且导致的均方根误差和压缩比为

```
> f3 = jpeg2k2im(c3);
>> rms3 = compare(f, f3)

rms3 =
    1.1234

>> cr3 = imratio(f, c3)
cr3 =
    1.6350
```

小结

本章介绍了通过消除编码冗余、像素间冗余和心理视觉冗余来进行数字图像压缩的基础知识，开发了解决这些冗余的MATLAB子程序，扩充了图像处理工具箱。最后，简要介绍了流行的JPEG和JPEG 2000图像压缩标准。关于消除图像冗余的其他信息，即本文未涉及的技术和特定图像子集（如二值图像）的标准，请参阅Gonzalez and Woods[2002]所著的《数字图像处理》一书的第8章。

第9章　形态学图像处理

前言

形态学一词通常指生物学的一个分支，它用于处理动物和植物的形状和结构。我们在数学形态学的语境中也使用该词来作为提取图像分量的一种工具，这些分量在表示和描述区域形状（如边界、骨骼和凸壳）时是很有用的。此外，我们还很关注用于预处理和后处理的形态学技术，如形态学滤波、细化和裁剪。

在9.1节中，我们将定义一些集合论运算，介绍二值图像，并讨论二值集合和逻辑运算符。在9.2节中，我们将根据图像平移形式（结构元素）的并集（或交集）定义两种基本的形态学运算——膨胀和腐蚀。在9.3节中，我们将组合处理膨胀和腐蚀，以便获得更为复杂的形态学运算。在9.4节中，我们将介绍在图像中标明连接部分的技术。这是我们从图像中提取目标以进行后续分析的基本步骤。

在9.5节中，我们将讨论形态学重构。一个形态学变换涉及到两幅图像，而不是一幅图像或一个结构元素，正如9.1节到9.4节中介绍的那样。在9.6节中，我们将通过用最大值和最小值代替集合的并集和交集。来把形态学的概念扩展到灰度图像。许多二值形态学运算可自然地推广到灰度图像的处理。有些像形态学重构这样的运算具有只有灰度图像才有的应用，如峰值滤波。

本章从图像处理方法开始过渡，这些方法中的输入和输出均为图像，对图像分析方法来说，其输出是以某种方法描述图像的内容。其他方法将在本书余下的章节中加以介绍和应用。

9.1　预备知识

在这一节中，我们将介绍一些集合理论中的基本概念，并讨论MATLAB的逻辑运算符对于二值图像的应用。

9.1.1　集合论中的基本概念

令Z为整数集合。用于产生数字图像的取样处理可以看成是把xy平面分隔成网格的处理，其中每个网格的中心坐标是来自笛卡儿乘积[1]Z^2中的一对元素。在集合论的术语中，若(x, y)是Z^2中的整数，f是为每对不同的坐标(x, y)分配亮度值（即源于实数集R中的实数）的映射，则函数$f(x, y)$称为数字图像。若R中的元素也是整数（本书中通常是这种情况），则该数字图像就变成了一个二维函数，它的坐标与振幅（即亮度）值均为整数。

令A为Z^2中的一个集合，其中的元素是像素坐标(x, y)。若$w = (x, y)$是A的一个元素，则我们可以写为

$$w \in A$$

同样，若w不是A的元素，则我们可以写为

---

① 一个整数集的笛卡儿乘积Z是元素(z_i, z_j)的所有序对的集合，z_i和z_j是来自Z的整数。通常用Z^2来定义这个集合。

9.2　膨胀和腐蚀

膨胀和腐蚀运算是形态学图像处理的基础。本章后面介绍的许多算法均基于这些运算，我们将在下面的讨论中定义并说明这些算法。

9.2.1　膨胀

膨胀是在二值图像中"加长"或"变粗"的操作。这种特殊的方式和变粗的程度由一个称为结构元素的集合控制。图9.4说明了膨胀的计算过程。图9.4(a)显示了包含一个矩形对象的简单二值图像；图9.4(b)是一个结构元素，在此例中它是一条5个像素长的斜线。计算时，结构元素通常用0和1的矩阵表示；有时，如图中所示，为方便起见可只显示1。另外，结构元素的原点必须明确标明。图9.4(b)用黑色方框标明了结构元素的原点。图9.4(c)明确地描述了膨胀处理，这种处理会将结构元素的原点平移过整个图像区域，并且核对哪些地方与值为1的像素重叠。图9.4(d)所示的输出图像在原点的每个位置均为1，从而在输入图像中结构元素至少重叠了一个1值像素。

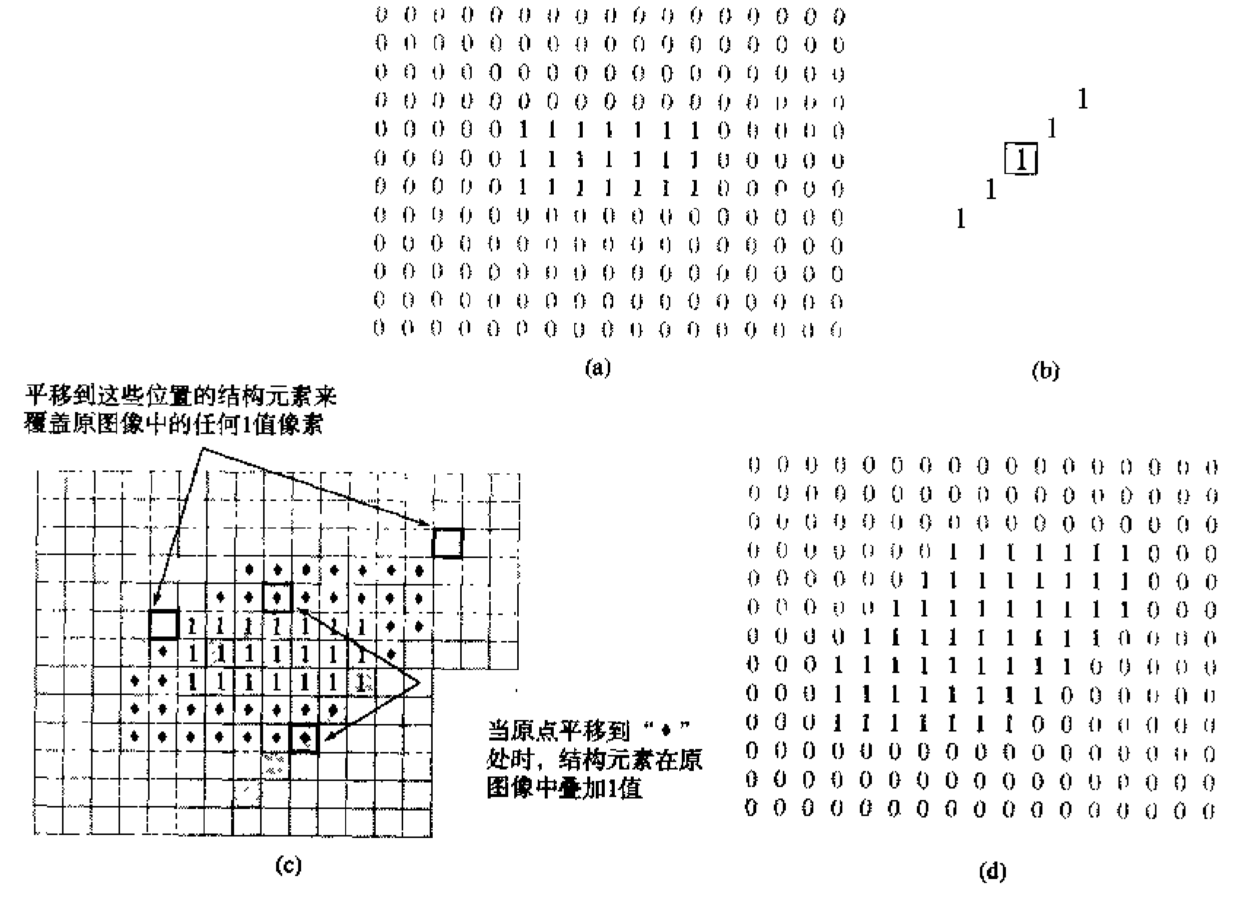


图9.4　膨胀示例：(a)带有矩形对象的原图像；(b)以对角线排列的有5个像素的结构元素，结构元素的原点带有黑框；(c)平移到几个位置后的结构元素；(d)输出图像

数学上，膨胀定义为集合运算。A被B膨胀，记为$A\oplus B$，定义为

$$A\oplus B=\{z\,|\,(\hat{B})_z\cap A\neq\varnothing\}$$

其中，$\varnothing$为空集，B为结构元素。总之，A被B膨胀是所有结构元素原点位置组成的集合，其中映射并平移后的B至少与A的某些部分重叠。这种在膨胀过程中对结构元素的平移类似于第3章中讨论

过的空间卷积。图 9.4 中并没有明显地显示出结构元素的映射，因为这种情况下结构元素是关于原点对称的。图 9.5 显示了非对称结构元素及其映射。

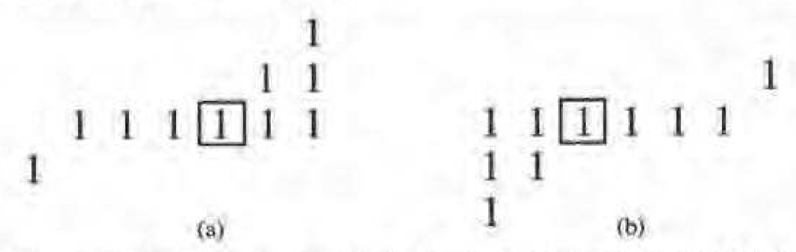


图 9.5　结构元素映像：(a)非对称结构元素；(b)结构元素基于原点的映像

膨胀满足交换律，即 $A \oplus B = B \oplus A$。在图像处理中，我们习惯于令 $A \oplus B$ 的第一个操作数为图像，而第二个操作数为结构元素，结构元素往往比图像小得多。从现在开始，我们将遵循这个原则。

例 9.1　膨胀的一个简单应用

IPT 函数 `imdilate` 执行膨胀运算。其基本的调用语法为

```
A2 = imdilate(A, B)
```

其中，A 和 A2 都是二值图像，B 是指定结构元素的由 0 和 1 组成的矩阵。图 9.6(a)显示了一幅样本二值图像，图像中包含有字符残缺的文本。我们想使用函数 `imdilate` 来膨胀具有以下结构元素的图像：

$$\begin{matrix} 0 & 1 & 0 \\ 1 & \boxed{1} & 1 \\ 0 & 1 & 0 \end{matrix}$$

下面的命令从一个文件中读取图像，形成结构元素矩阵，执行膨胀运算并显示结果。

```
>> A = imread('broken_text.tif');
>> B = [0 1 0; 1 1 1; 0 1 0];
>> A2 = imdilate(A, B);
>> imshow(A2)
```

图 9.6(b)显示了结果图像。

Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.

(a)

Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.

(b)

图 9.6　一个膨胀的简单例子：(a)包括残缺文本的输入图像；(b)膨胀后的图像

9.2.2　结构元素的分解

膨胀满足结合律，即

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C$$

假设假设一个结构元素 B 可以表示为两个结构元素 B_1 和 B_2 的膨胀，即

$$B = B_1 \oplus B_2$$

则 $A \oplus B = A \oplus (B_1 \oplus B_2) = (A \oplus B_1) \oplus B_2$，换言之，用 B 膨胀 A 等同于用 B_1 先膨胀 A，再用 B_2 膨胀前面的结果。我们称 B 能够分解成 B_1 和 B_2 两个结构元素。

结合律很重要，因为计算膨胀所需要的时间正比于结构元素中的非零像素的个数。例如，试考虑一个大小为 5 × 5 且其元素为 1 的数组膨胀：

$$\begin{matrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & \boxed{1} & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{matrix}$$

这个结构元素可分解为值一个为 1 的五元素行矩阵和一个值为 1 的五元素列矩阵。

$$[1 \; 1 \; \boxed{1} \; 1 \; 1] \oplus \begin{bmatrix} 1 \\ 1 \\ \boxed{1} \\ 1 \\ 1 \end{bmatrix}$$

原始结构元素中的元素个数为 25，但在行-列分解中的总元素数只有 10 个，这意味着首先用行结构元素膨胀，然后再用列结构元素膨胀，能够比 5 × 5 的数组膨胀快 2.5 倍。在实际中，速度多少会慢一些，因为在每个膨胀运算中总有些其他开销，至少在用分解形式时需要两次膨胀运算。然而，分解的实现所获得的速度增长仍然有很大意义。

9.2.3　函数 strel

IPT 函数 `strel` 运用各种形状和大小构造结构元素，其基本语法为

```
se = strel(shape, parameters)
```

其中，`shape` 是指定希望形状的字符串，而 `parameters` 是指定形状信息（如其大小）的一列参数。例如，`strel('diamond',5)` 将返回一个沿水平和垂直轴扩展±5个像素的菱形结构元素。表 9.2 总结了函数 `strel` 可以构造的各种形状。

表 9.2　函数 strel 的各种语法形式（"平坦的"一词意味着结构元素的高度为零，它只对灰度级膨胀和腐蚀有意义，参见 9.6.1 节）

语句形式	描述
`se = strel('diamond',R)`	创建一个平坦的菱形结构元素，其中 R 是从结构元素原点到菱形的最远点的距离
`se = strel('disk',R)`	创建一个平坦的圆盘型结构元素，其半径为 R（可为圆盘形指定其他的参数，详见 strel 的帮助页）
`se = strel('line',LEN,DEG)`	创建一个平坦的线性结构元素，其中 LEN 表示长度，DEG 表示线的角度（从水平轴起逆时针方向度量）
`se = strel('octagon',R)`	创建一个平坦的八边形结构元素，其中 R 是从结构元素原点到八边形的边的距离，沿水平轴和垂直轴度量。R 必须是 3 的非负倍数
`se = strel('pair',OFFSET)`	创建一个包含有两个成员的平坦结构元素，一个成员在原点，另一个成员的位置由向量 OFFSET 表示，该向量必须是一个两元素的整数向量

（续表）

语句形式	描述
se=strel('periodicline',P,V)	创建一个包含有2*p+1个成员的平坦结构元素。其中V是一个两元素向量，它包含有整数值的行和列偏移。一个结构元素成员在原点，其他成员位于1*V,-1*V,2*V,-2*V,...,P*V和-P*V
se=strel('rectangle',MN)	创建一个平坦的矩形结构元素，其中MN指定了大小。MN必须是一个两元素的非负整数向量。MN中的第一个元素是结构元素中的行数，第二个元素是列数
se=strel('square',W)	创建一个方形的结构元素，其边长为W个像素。W必须是一个非负整数标量
se=strel('arbitrary',NHOOD) se=strel(NHOOD)	创建一个任意形状的结构元素。NHOOD是由0和1组成的矩阵，用于指定形状。显示的第二种更简单的语法形式执行相同的操作

除了可简化常用结构元素形状的产生之外，函数strel还有一个重要的属性，即以分解的形式来产生结构元素。函数imdilate自动地使用分解信息来加快膨胀处理。以下例子说明了strel如何返回与结构元素的分解相关的信息。

例9.2　使用函数strel分解结构元素的说明

再次考虑使用函数strel来创建一个菱形的结构元素：

```
>> se = strel('diamond', 5)
se =
Flat STREL object containing 61 neighbors.
Decomposition: 4 STREL objects containing a total of 17 neighbors
Neighborhood:
     0     0     0     0     0     1     0     0     0     0     0
     0     0     0     0     1     1     1     0     0     0     0
     0     0     0     1     1     1     1     1     0     0     0
     0     0     1     1     1     1     1     1     1     0     0
     0     1     1     1     1     1     1     1     1     1     0
     1     1     1     1     1     1     1     1     1     1     1
     0     1     1     1     1     1     1     1     1     1     0
     0     0     1     1     1     1     1     1     1     0     0
     0     0     0     1     1     1     1     1     0     0     0
     0     0     0     0     1     1     1     0     0     0     0
     0     0     0     0     0     1     0     0     0     0     0
```

我们看到函数strel并不显示为通常的MATLAB矩阵，而是返回一个称为strel对象的特殊量。命令窗口显示出的strel对象包括其邻域（本例中为元素为1的菱形矩阵）、结构元素中值为1的像素数(61)、分解中的结构元素数(4)以及分解的结构元素中值为1的总像素数(17)。函数getsequence可用于提取并检查分解中的单个结构元素。

```
>> decomp = getsequence(se);
>> whos
  Name        Size                Bytes  Class
  decomp      4 x 1                1716  strel object
  se          1 x 1                3309  strel object
Grand total is 495 elements using 5025 bytes
```

whos的输出表明se和decomp均为strel对象，进一步说，decomp是strel对象的四元素向量。分解中的四结构元素可通过索引到decomp中来逐个地检查。

```
>> decomp(1)
ans =
```

```
Flat STREL object containing 5 neighbors.
Neighborhood:
     0     1     0
     1     1     1
     0     1     0

>> decomp(2)

ans =

Flat STREL object containing 4 neighbors.

Neighborhood:
     0     1     0
     1     0     1
     0     1     0

>> decomp(3)

ans =

Flat STREL object containing 4 neighbors.

Neighborhood:
     0     0     1     0     0
     0     0     0     0     0
     1     0     0     0     1
     0     0     0     0     0
     0     0     1     0     0

>> decomp(4)

ans =

Flat STREL object containing 4 neighbors.

Neighborhood:
     0     1     0
     1     0     1
     0     1     0
```

函数 imdilate 自动地使用结构元素的分解形式，执行膨胀运算的速度大约比非分解形式的速度快三倍（≈ 61/17）。

9.2.4 腐蚀

腐蚀“收缩”或“细化”二值图像中的对象。像在膨胀中一样，收缩的方式和程度由一个结构元素控制。图9.7说明了腐蚀过程。图9.7(a)和图9.4(a)相同，图9.7(b)是结构元素，即一条短垂直线。图9.7(c)生动地描述了腐蚀在整个图像区域平移结构元素的过程，并检查在哪里完全匹配图像的前景部分。图9.7(d)所示的输出图像中，结构元素原点的每个位置的值为1，因而该元素仅叠加了输入图像的1值像素（即它并不叠加任何图像背景）。

腐蚀的数学定义与膨胀相似，A 被 B 腐蚀记为$A \ominus B$，定义为

$$A \ominus B = \{z | (B)_z \cap A^c \neq \varnothing\}$$

换言之，A 被 B 腐蚀是所有结构元素的原点位置的集合，其中平移的 B 与 A 的背景并不叠加。

例9.3　腐蚀的说明

腐蚀用IPT函数 imerode 执行。假设我们要除去图9.8(a)中的细线，但我们想保留其他结构。通过选取一个足够小的结构元素来匹配中心方块，我们可实现这种处理；由于边框引线太粗，因而不能完全匹配这些线。考虑如下的命令：

```
>> A = imread('wirebond_mask.tif');
>> se = strel('disk', 10);
```

```
>> A2 = imerode(A, se);
>> imshow(A2)
```

如图 9.8(b)所示，这些命令成功地删除了掩模中的细线。图 9.8(c)显示了我们选择太小的结构元素时所发生的情形：

```
>> se = strel('disk', 5);
>> A3 = imerode(A, se);
>> imshow(A3)
```

在这种情形下，一些引线没未删除。图 9.8(d)显示了我们选择太大的结构元素时所发生的情形：

```
>> A4 = imerode(A, strel('disk', 20));
>> imshow(A4)
```

虽然这些引线已被删除，但边框引线也被删除了。

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

(a)

```
1
1
1
```

(b)

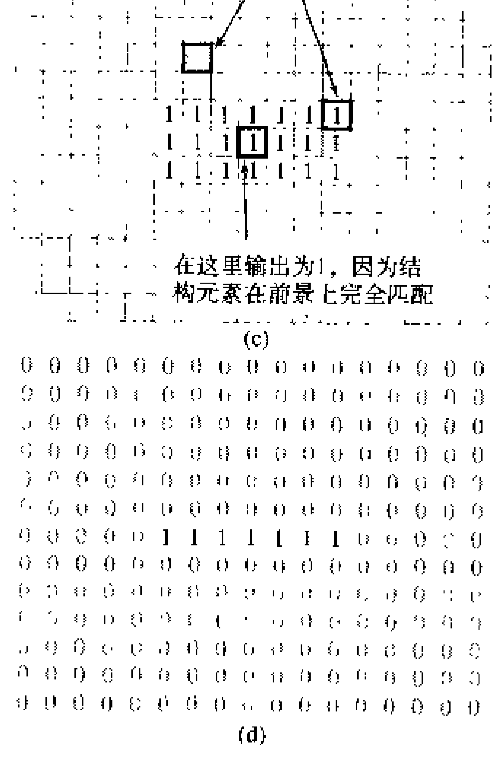


图 9.7 腐蚀的说明：(a)带有矩形对象的原图像；(b)垂直排列的 3 像素结构元素，结构元素的原点带有黑框；(c)被平移到图像上不同位置的结构元素；(d)输出图像

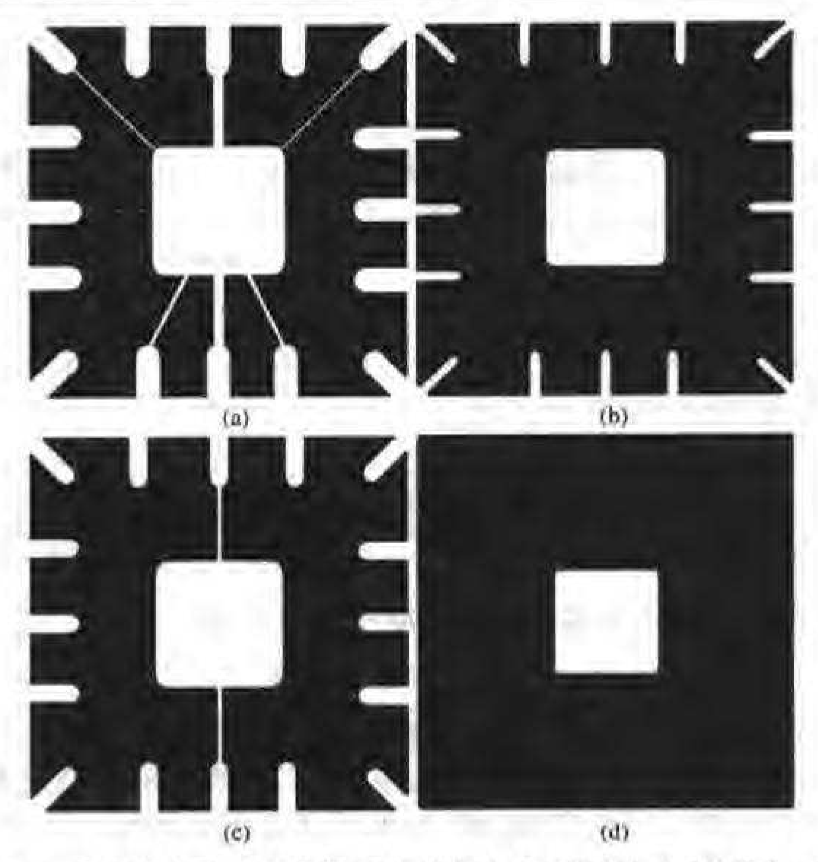


图 9.8 腐蚀的说明：(a)原图像；(b)用半径为10的圆盘腐蚀后的图像；(c)用半径为5的圆盘腐蚀后的图像；(d) 用半径为20的圆盘腐蚀后的图像

9.3 膨胀与腐蚀的组合

在图像处理的实际应用中，我们更多地以各种组合的形式来使用膨胀和腐蚀。一幅图像将使用相同或有时不同的结构元素来进行一系列膨胀或腐蚀运算。这一节考虑三种最常用的膨胀和腐蚀的组合：开运算、闭运算、击中或击不中变换。我们还将介绍查找表操作，并讨论IPT函数`bwmorph`，该函数能够执行多种实际的形态学任务。

9.3.1 开运算和闭运算

A 被 B 的形态学开运算可以记做$A \circ B$，这种运算是 A 被 B 腐蚀后再用 B 来膨胀腐蚀结果：

$$A \circ B = (A \ominus B) \oplus B$$

开运算的另一个数学公式为

$$A \circ B = \cup \{(B)_z | (B)_z \subseteq A\}$$

其中，$\cup\{\cdot\}$ 指大括号中所有集合的并集，符号 $C \subseteq D$ 表示C是D的一个子集。该公式的简单几何解释为：$A \circ B$是 B 在 A 内完全匹配的平移的并集。图 9.9 示例了这种解释。图 9.9(a)显示了集合 A 和圆盘形结构元素 B。图 9.9(b)显示了 B 在A 内完全匹配的一些平移。这些平移的并集为图 9.9(c)中的阴影部分；这个区域即为开运算的结果。该图形中的白色区域是结构元素不能完全在A中匹配的区域，因而不是开运算的结果部分。形态学开运算完全删除了不能包含结构元素的对象区域，平滑了对象的轮廓，断开了狭窄的连接，去掉了细小的突出部分。

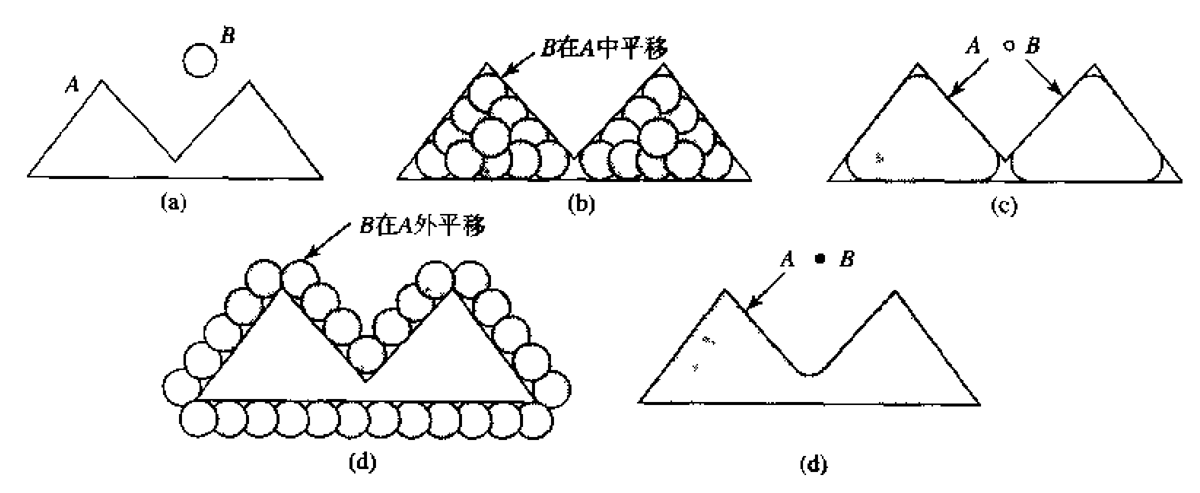


图 9.9　开运算和闭运算是平移的结构元素的并集：(a)集合 A 和结构元素 B；(b)在集合 A 内完全匹配的 B 的平移；(c)开运算的结果（阴影区）；(d)B 在框 A 外平移；(e)闭运算的结果（阴影区）

A 被 B 的形态学闭运算记做 $A \bullet B$，它是先膨胀再腐蚀的结果：

$$A \bullet B = (A \oplus B) \ominus B$$

从几何学上讲，$A \bullet B$ 是所有不与 A 重叠的 B 的平移的并集。图 9.9(d)显示了一些与 A 不重叠的 B 的平移，通过完成以上平移并集操作，我们得到了图 9.9(e)所示的阴影区，这就是闭运算的结果。像开运算一样，形态学闭运算会平滑对象的轮廓。然而，与开运算不同的是，闭运算一般会将狭窄的缺口连接起来形成细长的弯口，并填充比结构元素小的洞。

开运算和闭运算在工具箱中用函数 imopen 和 imclose 实现。这两个函数的语法形式为

```
C = imopen(A, B)
```

和

```
C = imclose(A, B)
```

其中，A 是一幅二值图像，而 B 是一个元素值为 0 和 1 的矩阵，该矩阵指定了结构元素。strel 对象 SE 可以用来代替 B。

例 9.4　函数 imopen 和 imclose 的应用

本例说明了函数 imopen 和 imclose 的应用。图 9.10(a)所示的图像 shapes.tif 有几个用于说明开运算和闭运算的特有结果的特性，例如细长的突出部分、细长的连线、弯口、孤立的小洞、小的孤立物和齿状边缘。使用 20 × 20 的结构元素对图像进行开运算的命令如下：

```
>> f = imread('shapes.tif');
>> se = strel('square', 20);
>> fo = imopen(f, se);
>> imshow(fo)
```

图 9.10(b)显示了结果。从中可以看出，细长的突出部分和指向外部的齿状边缘已被删除。细长的连线和小的孤立物也已被删除。命令

```
>> fc = imclose(f, se);
>> imshow(fc)
```

产生如图 9.10(c)所示的结果。这里，细长弯口、指向内部的齿状边缘和小洞都已被删除。正像下面将说明的那样，把开运算和闭运算组合起来使用将能够非常有效地去除噪声。根据图 9.10，

在先前开运算的结果上进行一次闭运算，最终非常成功地平滑了目标。对开运算后的图像进行闭运算的计算过程如下：

```
>> foc = imclose(fo, se);
>> imshow(foc)
```

图9.10(d)显示了平滑后的对象。

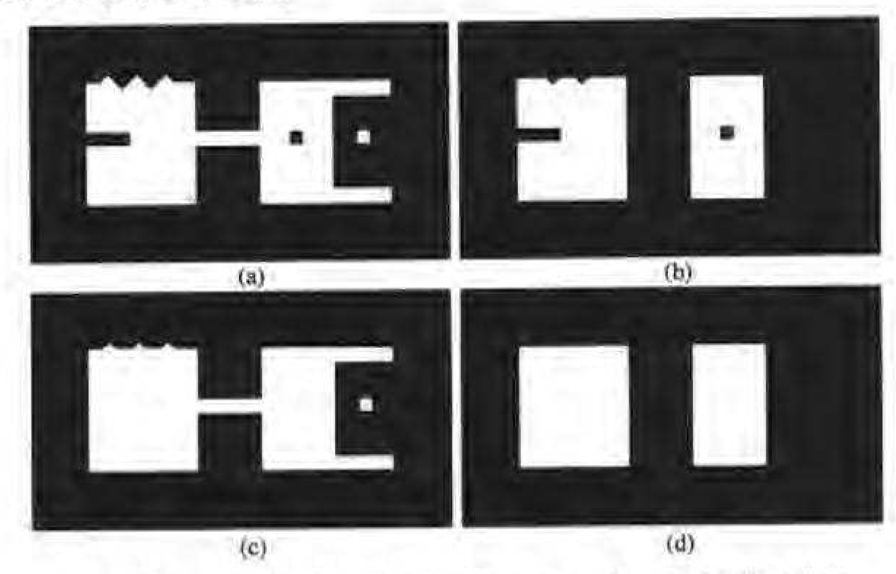


图9.10　开运算和闭运算的说明：(a)原图像；(b)开运算后的图像；(c)闭运算后的图像；(d)图像(b)经闭运算后的结果

图9.11通过对一幅存在杂散点的指纹图像应用这些运算，进一步说明了开运算和闭运算的用处。命令

```
>> f = imread('fingerprint.tif');
>> se = strel('square', 3);
>> fo = imopen(f, se);
>> imshow(fo)
```

产生了如图9.11(b)所示的图像。注意，对图像做开运算消除了这些杂散点，但该过程在纹脊上又引入了许多缺口。这些缺口可以通过开运算后再做闭运算来填充：

```
>> foc = imclose(fo,se);
>> imshow(foc)
```

图9.11(c)显示了最终的结果。



图9.11　(a)带有杂散点的指纹图像；(b)经开运算后的图像；(c)经开运算后再做闭运算所得到的图像（原图像由美国国家标准与技术研究所提供）

9.3.2 击中或击不中变换

通常，能够识别像素的特定形状是很有用的，例如孤立的前景像素或者是线段的端点像素。击中或击不中变换对这类应用非常实用（见图 9.12）。A 被 B 击中或击不中变换定义为 $A \otimes B$。其中，B 是结构元素对 $B = (B_1, B_2)$，而不是单个元素。击中或击不中变换由这两个结构元素定义为 $A \otimes B = (A \ominus B_1) \cap (A^c \ominus B_2)$。

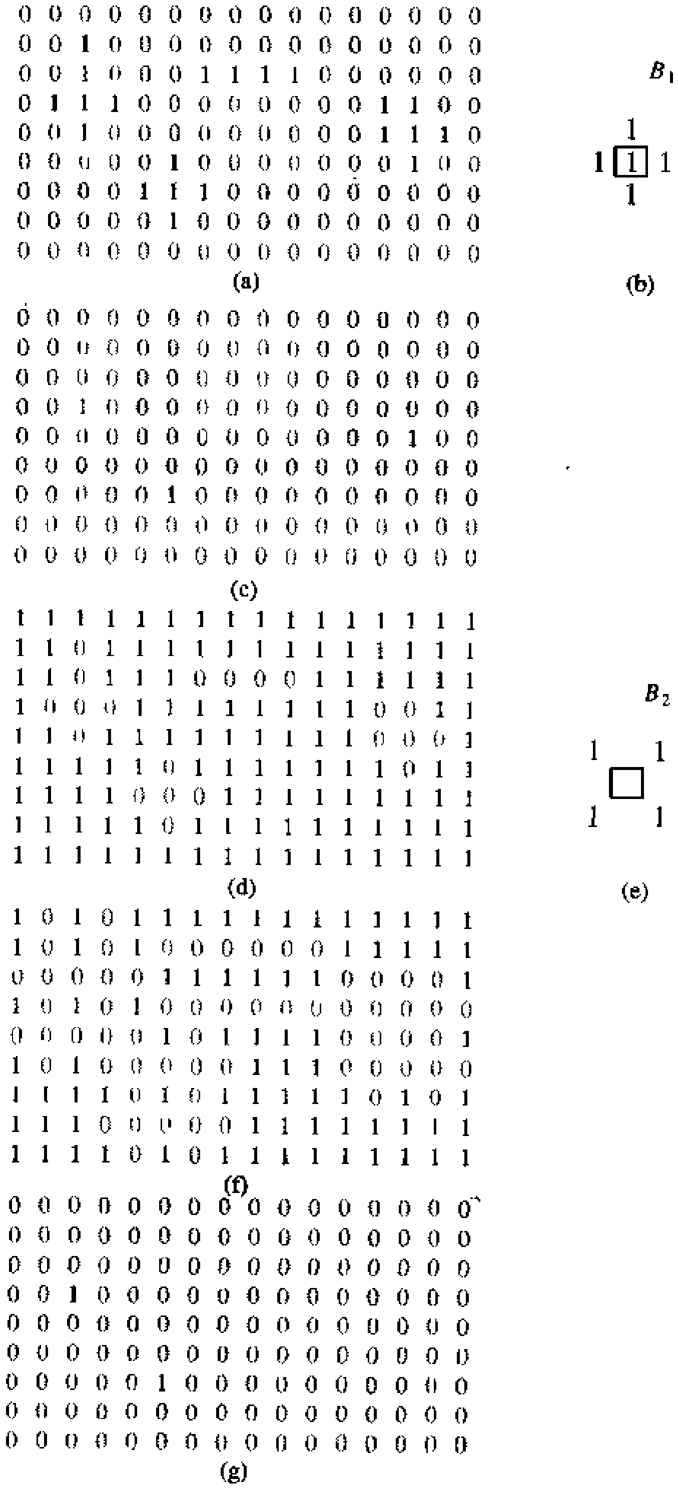


图 9.12 (a)原图像 A；(b)结构元素 B_1；(c)A 被 B_1 腐蚀；(d)原图像的补集 A^c；(e)结构元素 B_2；(f)A^c 被 B_2 腐蚀；(g)输出图像

9.3.3 使用查找表

当击中或击不中结构元素较小时，计算击中或击不中变换的较快方法是使用查找表（LUT）。这种方法是预先计算出每个可能邻域形状的像素值，然后把这些值存储到一个表中，以备以后使用。例如，在二值图像中有 $2^9 = 512$ 个不同的 3 × 3 像素值形状。要使用查找表，必须为每个可能的形状分配一个惟一的索引。一种简单的方法是用每个 3 × 3 形状的元素与下面的矩阵相乘：

$$\begin{matrix} 1 & 8 & 64 \\ 2 & 16 & 128 \\ 4 & 32 & 256 \end{matrix}$$

然后把所有的乘积加起来。该过程为每个3 × 3邻域形状在范围[0, 511]中分配一个惟一的值。例如，分配给邻域

$$\begin{matrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{matrix}$$

的值为 $1(1) + 2(1) + 4(1) + 8(1) + 16(0) + 32(0) + 64(0) + 128(1) + 256(1) = 399$，其中，积中第一个数字是来自前一个矩阵的系数，括号中的数字是像素值。

工具箱提供两个函数`makelut`和`applylut`（将在本节稍后说明），这两个函数可用于实现这种技术。函数`makelut`基于一个提供给用户的函数构造一个查找表，函数`applylut`则使用这个查找表来处理二值图像。仍考虑上面的3 × 3情形，使用函数`makelut`要求写一个接受3 × 3二值矩阵并返回一个单值的函数，其典型值不是0就是1。函数`makelut`通过每个可能的3 × 3邻域调用用户函数512次。它以512个元素向量的形式记录并返回所有的值。

作为示例，我们写一个函数`endpoints.m`，该函数使用`makelut`和`applylut`在二值图像中检测端点。我们定义一个端点作为前景像素，它有一个确切的前景邻域。函数`endpoints`计算并应用查找表在输入图像中检测端点。函数`endpoints`中使用的代码行

```
persistent lut
```

建立一个名为`lut`的变量并将该变量声明为持久的。MATLAB会在两次函数调用间“记住”该持久变量的值。第一次调用函数`endpoints`时，变量`lut`会自动初始化为空矩阵([ ])。当`lut`为空时，该函数调用`makelut`，子函数`endpoint_fcn`的句柄将作为`makelut`的参量。函数`applylut`然后使用查找表找到端点。查找表存储在持久变量`lut`中，以便在下一次调用`endpoints`时无须重新计算查找表。

```
function g = endpoints(f)
%ENDPOINTS Computes end points of a binary image.
%   G = ENDPOINTS(F) computes the end points of the binary image F
%   and returns them in the binary image G.

persistent lut

if isempty(lut)
   lut = makelut(@endpoint_fcn, 3);
end

g = applylut(f, lut);

%-------------------------------------------------------------------%
function is_end_point = endpoint_fcn(nhood)
%   Determines if a pixel is an end point.
```

```
%    IS_END_POINT = ENDPOINT_FCN(NHOOD) accepts a 3-by-3 binary
%    neighborhood, NHOOD, and returns a 1 if the center element is an
%    end point; otherwise it returns a 0.

is_end_point = nhood(2, 2) & (sum(nhood(:)) == 2);
```

图9.14示例了函数endpoints的典型用途。图9.14(a)是一幅包含有形态学骨骼的二值图像（见9.3.4节）。图9.14(b)显示了函数endpoints的输出。

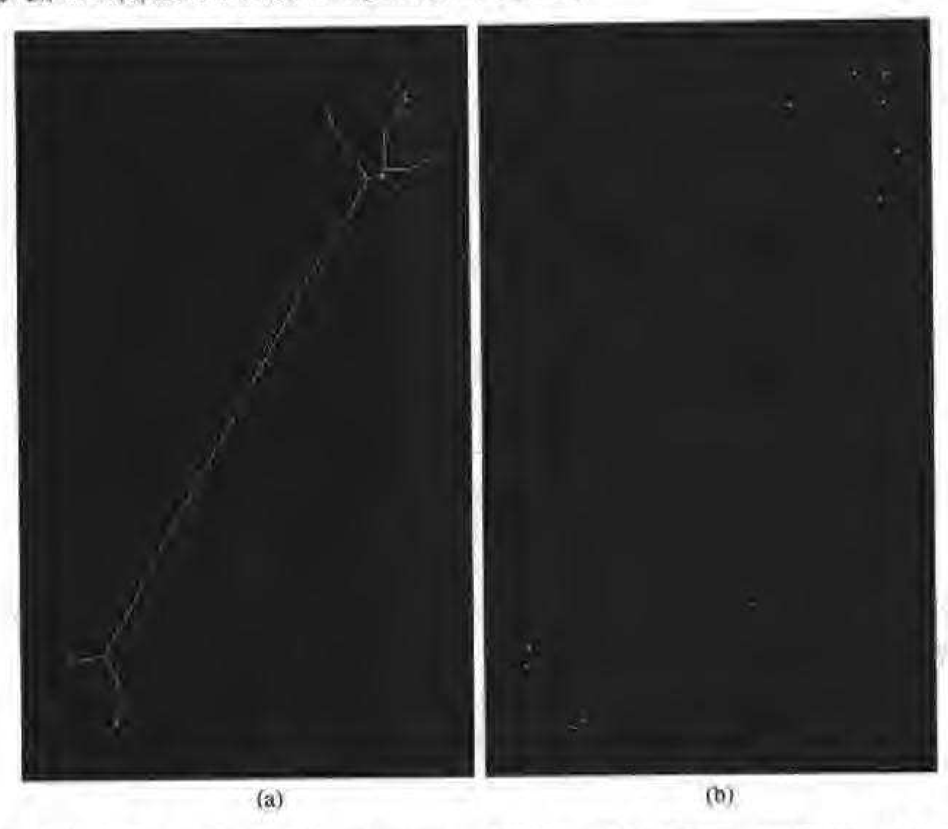


图9.14　(a)形态骨骼的图像；(b)函数endpoints的输出。为清楚起见，图像(b)中的像素已被放大

例9.6　使用二值图像和基于查找表的计算玩Conway的生命游戏

查找表的一种有趣应用是Conway的“生命游戏”。该游戏中包含有排列在矩形网格上的许多“生物体”。举这个例子的目的在于说明查找表的强大功能和简便性。在Conway的游戏中，有一些简单的规则决定这些生物体如何产生、生存、死亡和繁衍。我们可用一幅二值图像来方便地表述该游戏，图像中的每个前景像素表示该位置上的一个生物体。

Conway的遗传规则描述了如何从现有的一代生物体来计算下一代生物体（或下一幅二值图像）：

1. 每个拥有两个或者三个相邻前景像素的前景像素可以产生下一代。
2. 每个有零个、一个或至少四个相邻前景像素的前景像素会“死亡”，因为这种前景像素要么被“孤立”，要么“数目过多”。
3. 与三个前景邻域相连的每个背景像素是一个“新生”像素，并且会变成一个前景像素。

在计算描述下一代的下一幅二值图像的过程中，所有的新生和死亡会同时发生。

要使用makelut和applylut来实现该生命游戏，我们首先要写一个函数，该函数会为一个像素及其3 × 3邻域应用Conway的遗传规则：

```
function out = conwaylaws(nhood)
```

```
%CONWAYLAWS Applies Conway's genetic laws to a single pixel.
%   OUT = CONWAYLAWS(NHOOD) applies Conway's genetic laws to a single
%   pixel and its 3-by-3 neighborhood, NHOOD.

num_neighbors = sum(nhood(:)) - nhood(2, 2);
if nhood(2, 2) == 1
   if num_neighbors <= 1
      out = 0; % Pixel dies from isolation.
   elseif num_neighbors > = 4
      out = 0; % Pixel dies from overpopulation.
   else
      out = 1; % Pixel survives.
    end
else
    if num_neighbors == 3
       out = 1; % Birth pixel.
    else
       out = 0; % Pixel remains empty.
    end
end
```

通过使用函数 conwaylaws 的句柄来调用 makelut，可构建查找表：

```
>> lut = makelut(@conwaylaws, 3);
```

各种初始图像已被设计好，以便演示 Conway 规则对后续代的影响（见 Gardner[1970, 1971]）。例如，考虑称为"大笑猫"的初始图像：

```
>> bw1 = [0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0
          0 0 0 1 0 0 1 0 0 0
          0 0 0 1 1 1 1 0 0 0
          0 0 1 0 0 0 0 1 0 0
          0 0 1 0 1 1 0 1 0 0
          0 0 1 0 0 0 0 1 0 0
          0 0 0 1 1 1 1 0 0 0
          0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0 0];
```

下面的命令执行计算并显示到第三代：

```
>> imshow(bw1, 'n'), title('Generation 1')
>> bw2 = applylut(bw1, lut);
>> figure, imshow(bw2, 'n'); title('Generation 2')
>> bw3 = applylut(bw2, lut);
>> figure, imshow(bw3, 'n'); title('Generation 3')
```

我们将把它作为一个练习，以显示经过几代以后猫大笑前的微笑情况。

9.3.4 函数 bwmorph

IPT 函数 bwmorph 可基于膨胀、腐蚀和查找表操作的组合实现许多有用的操作，该函数的调用语法为

对象相关联的呢？虽然它们以两个单独的组的形式出现，但所有16个像素实际上都属于图9.17(a)中的字母"E"。为开发定位和操作对象的计算机程序，我们需要为关键术语做一系列更为精确的定义。

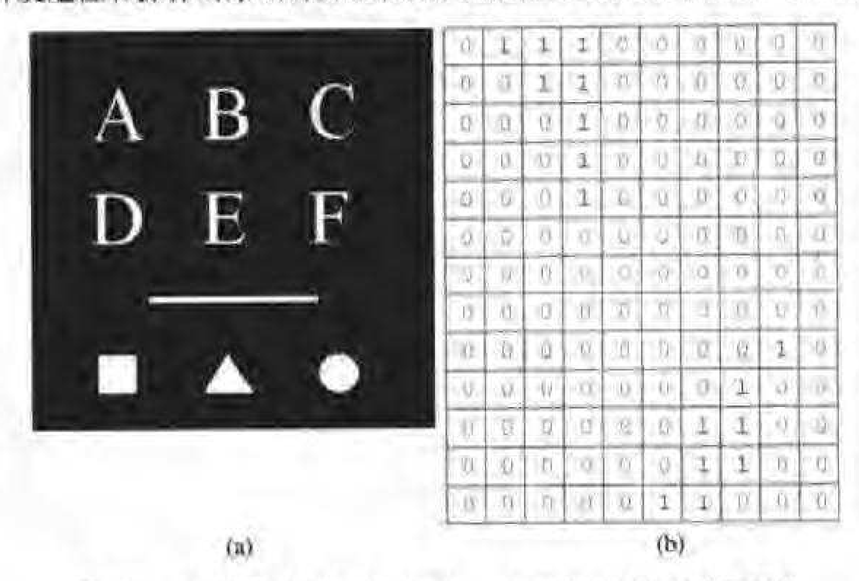


图9.17　(a) 包含10个物体的图像；(b) 来自该图像的像素子集

一个坐标为(x, y)的像素p有两个水平和两个垂直的相邻像素，它们的坐标分别为$(x+1, y)$，$(x-1, y)$，$(x, y+1)$和$(x, y-1)$。p的这4个相邻像素的集合记为$N_4(p)$，即图9.18(a)中的阴影部分。p的4个对角线相邻像素的坐标分别为$(x+1, y+1)$，$(x+1, y-1)$，$(x-1, y+1)$和$(x-1, y-1)$。图9.18(b)显示了这些相邻像素，它们被记为$N_D(p)$。而图9.18(c)中的$N_4(p)$和$N_D(p)$的并集是p的8个相邻像素，记为$N_8(p)$。

若$q \in N_4(p)$，则像素p和q称为4邻接。同样，若$q \in N_8(p)$，则p和q称为8邻接。图9.18(d)和图9.18(e)说明了这些概念。p_1和p_n之间的一条路径是一系列像素p_1，p_2，…，p_{n-1}，p_n。其中p_k和p_{k+1}相邻，$1 \leq k < n$。一条路径可以是4连接，也可以是8连接，具体取决于所用邻接的定义。

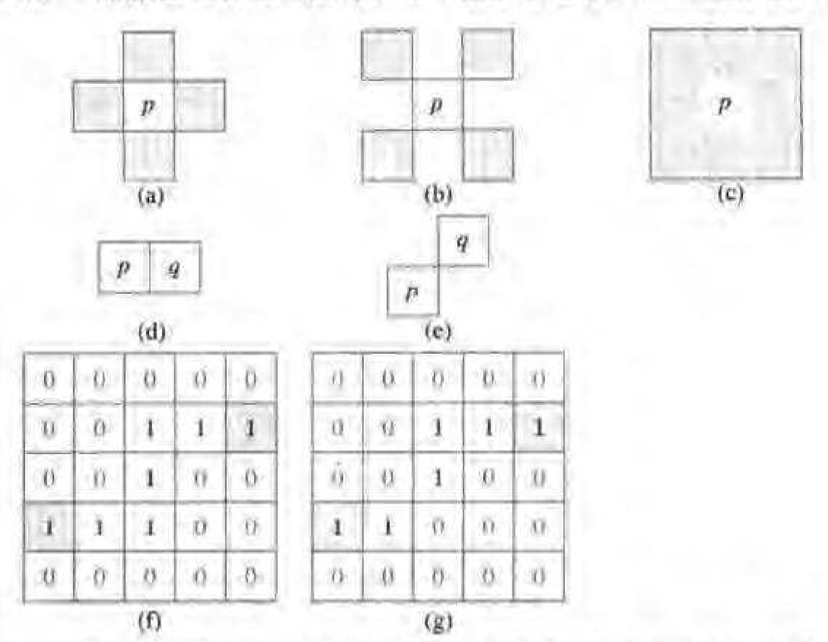


图9.18　(a)像素p及其4个相邻像素$N_4(p)$；(b)像素p及其对角相邻像素$N_D(p)$；(c)像素p及其8个相邻像素$N_8(p)$；(d)像素p和q是4邻接和8邻接；(e)像素p和q是8邻接而不是4邻接；(f)阴影像素既是4连接像素又是8连接像素；(g)阴影前景像素是8连接像素而不是4连接像素

若在前景像素p和q之间存在一条完全由前景像素组成的4连接路径，则这两个前景像素称为4连接［见图9.18(f)］。若它们之间存在一条8连接路径，则称为8连接［见图9.18(g)］。对于任意前景像素p，与其相连的所有前景像素的集合称为包含p的连接分量。

连接分量这一术语是根据路径来定义的，而路径的定义则取决于邻接。这就表明连接分量的性质取决于我们所选的邻接方式，最常见邻接方式为4邻接和8邻接。图9.19说明了邻接方式对确定图像中连接分量的数量的影响。图9.19(a)显示了具有四个4连接分量的一幅小二值图像，图9.19(b)显示了选择8邻接可将连接分量的数量减少为两个。

IPT 函数 `bwlabel` 可用于计算一幅二值图像中的所有连接分量。调用语法为

```
[L, num] = bwlabel(f, conn)
```

其中，`f` 是一幅输入二值图像，`conn` 用于指定期望的连接（不是4就是8）。输出 `L` 称为标记矩阵，参数 `num`（可选）给出所找到的连接分量的总数。若省略了参数 `conn`，则其值默认为8。图9.19(c)显示了与图9.19(a)相对应的标记矩阵，该矩阵是使用函数 `bwlabel(f,4)` 计算得到的。每个不同连接分量中的像素被分配给一个惟一的整数，该整数的范围是从1到连接分量的总数。换言之，标记值为1的像素属于第一个连接分量；标记值为2的像素属于第二个连接分量；依次类推。背景像素标记为0。图9.19(d)显示了与图9.19(a)相对应的标记矩阵，该矩阵是用函数 `bwlabel(f,8)` 计算得到的。

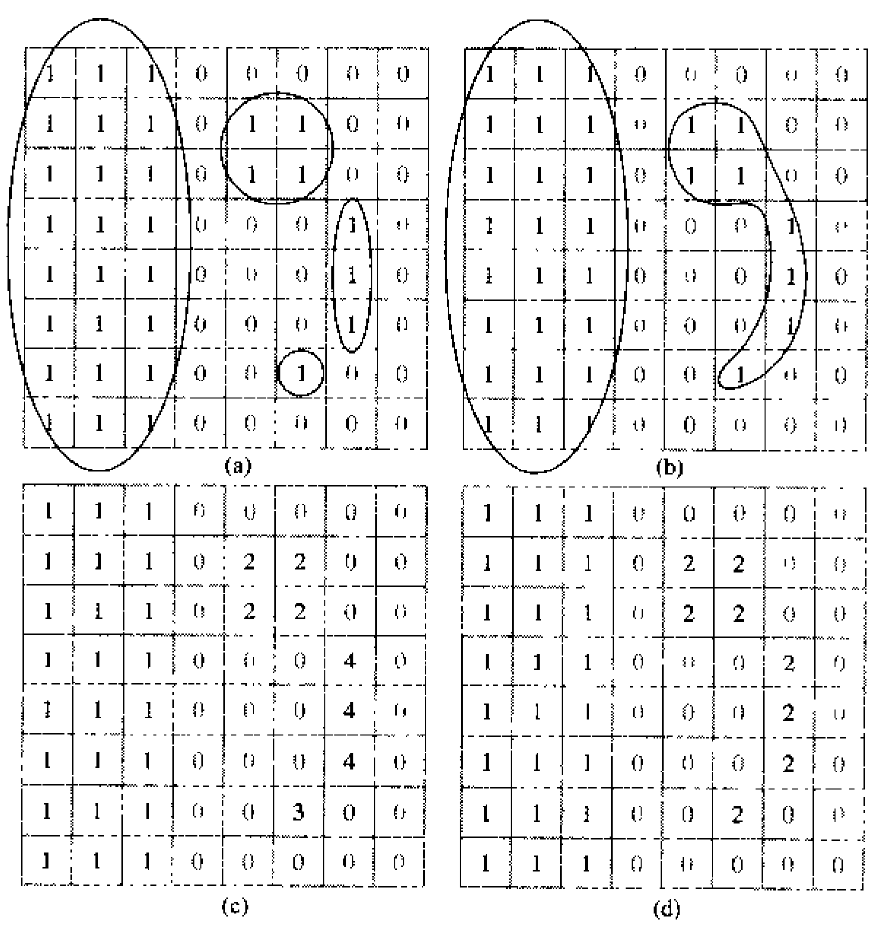


图9.19 连接分量：(a)四个4连接分量；(b)两个8连接分量；(c)使用4连接得到的标记矩阵；(d)使用8连接得到的标记矩阵

例9.7 计算和显示连接分量的质心

本例显示了如何计算和显示图9.17(a)中的连接分量的质心。首先，我们使用函数 `bwlabel` 来计算8连接分量：

```
>> f = imread('objects.tif');
>> [L, n] = bwlabel(f);
```

函数 find（见5.2.2节）在处理标记矩阵时非常有用。例如，以下对 find 的调用将返回属于第三个对象的所有像素的行索引和列索引：

```
>> [r, c] = find(L == 3);
```

然后，利用 r 和 c 作为输入的 mean 函数来计算该对象的质心。

```
>> rbar = mean(r);
>> cbar = mean(c);
```

可以使用一个循环来计算和显示图像中全部对象的质心。为了使质心叠置在图像上时可看到该质心，我们将使用中心为白色“*”符号的黑色圆标记来表示，如下所示：

```
>> imshow(f)
>> hold on    % So later plotting commands plot on top of the image.
>> for k = 1:n
   [r, c] = find(L == k);
   rbar = mean(r);
   cbar = mean(c);
   plot(cbar, rbar, 'Marker', 'o', 'MarkerEdgeColor', 'k',...
        'MarkerFaceColor', 'k', 'MarkerSize', 10);
   plot(cbar, rbar, 'Marker', '*', 'MarkerEdgeColor', 'w');
end
```

图9.20显示了结果。



图9.20　叠置在相应连接分量上的质心（白色星号）

9.5　形态学重构

重构是一种涉及到两幅图像和一个结构元素（而不是单幅图像和一个结构元素）的形态学变换。一幅图像，即标记（marker），是变换的开始点。另一幅图像是掩模（mask），用来约束变换过程。结构元素用于定义连接性。在本节中，我们将使用8连接（默认值），这表明如下讨论中的B是一个大小为3 × 3且值为1的矩阵，其中心坐标为(2, 2)。

若 g 是掩模，f 为标记，则从 f 重构 g 可以记为 $R_g(f)$，它由下面的迭代过程定义：

1. 将 h_1 初始化为标记图像 f。
2. 创建结构元素：`B = ones(3)`。
3. 重复

$$h_{k+1} = (h_k \oplus B) \cap g$$

直到 $h_{k+1} = h_k$

标记 f 必须是 g 的一个子集，即

$$f \subseteq g$$

图 9.21 说明了上述迭代过程。注意，虽然这个迭代公式在概念上很有用，但还存在更为快速的计算方法。IPT 函数 `imreconstruct` 使用的是在 Vincent[1993] 中描述的“快速混合重构”算法。函数 `imreconstruct` 的调用语法为

```
out = imreconstruct(marker, mask)
```

其中，`marker` 和 `mask` 的定义见本节的开始之处。

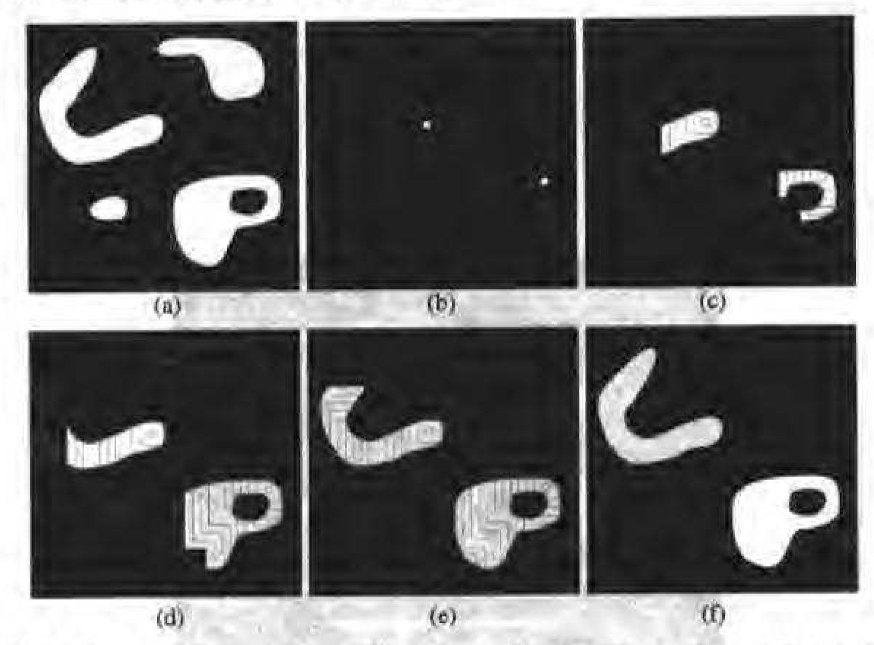


图 9.21　形态学重构：(a)原图像；(b)标记图像；(c)~(e) 分别经 100 次、200 次和 300 次迭代后的中间结果；(f)最终结果［为便于查看，掩模图像中对象的外轮廓已叠置在图(b)~(e)上］

9.5.1　由重构做开运算

在形态学开运算中，腐蚀通常会去除小的对象，而随后的膨胀往往会还原所保留对象的形状。然而，这种还原的精度取决于形状和结构元素之间的相似性。本节所讨论的方法，即由重构做开运算，将准确地恢复腐蚀之后的对象形状。使用结构元素 B 对 f 由重构做开运算的定义为 $R_f(f \ominus B)$。

例 9.8　由重构做开运算

图 9.22 对包含文字的图像进行了开运算与由重构做开运算之间的比较。在这个例子中，我们的兴趣在于从图 9.22(a)中提取含有长竖线的文字。由于由重构做开运算需要一幅被腐蚀的图像，所以我们首先使用一个竖向的细结构元素，该元素的长度与符号的高度成比例：

```
>> f = imread('book_text_bw.tif');
>> fe = imerode(f, ones(51, 1));
```

图9.22(b)显示了结果。图9.22(c)中所示的开运算使用函数 imopen 来计算：

```
>> fo = imopen(f,ones(51, 1));
```

注意，竖条已被复原，但未包括字符中除竖条外的其余部分。最后，我们获得了重构：

```
>> fobr = imreconstruct(fe, f);
```

图9.22(d)中的结果完全地复原了含有长竖条的字符，而其他字符则已被去掉。图9.22中的其余两幅图像将在以下两小节中解释。

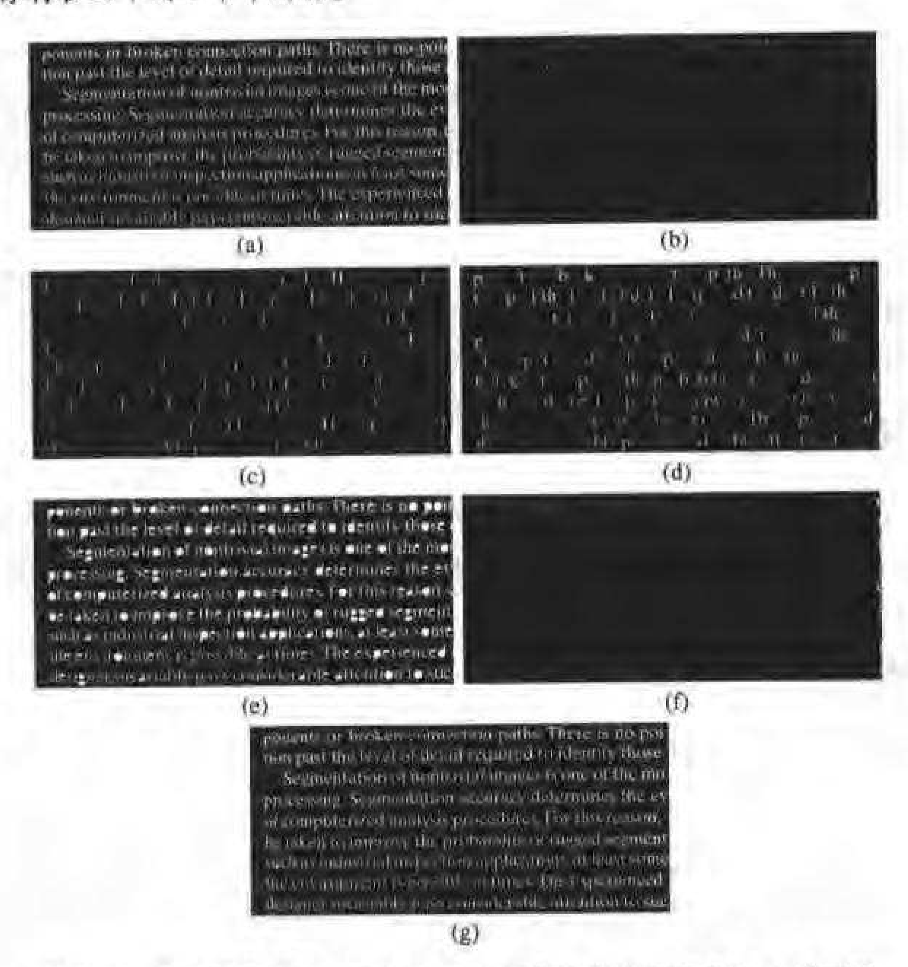


图9.22 形态学重构：(a)原图像；(b)使用竖线腐蚀后的图像；(c)使用竖线做开运算后的结果；(d)使用竖线由重构做开运算后的结果；(e)填充的孔洞；(f)接触边界的字符；(g)删除边界字符后的图像

9.5.2 填充孔洞

形态学重构的应用范围很广，每种应用都取决于标记和掩模图像的选择。例如，假设我们选择了一幅标记图像f_m，该图像的边缘部分的值为$1-f$，其余部分的值为0：

$$f_m(x,y)=\begin{cases}1-f(x,y) & 若(x,y)在f的边界上\\ 0 & 其他\end{cases}$$

$$b(x', y') = 0, \qquad (x', y') \in D_b$$

在这种情况下，最大值运算完全由二值矩阵 D_b 中的 0 和 1 模式来指定，且灰度膨胀公式可简化为

$$(f \oplus b)(x, y) = \max\{f(x - x', y - y') \mid (x', y') \in D_b\}$$

因此，平坦的灰度膨胀是一个局部最大值算子，其中的最大值取自由 D_b 的形状所确定的一系列像素邻域。

非平坦的结构元素可通过函数 strel 来创建，该函数的参量是两个矩阵：(1)指定结构元素的定义域 D_b 的矩阵，其元素值为 0 和 1；(2)指定高度值 $b(x', y')$ 的第二个矩阵。例如，

```
>> b = strel([1 1 1], [1 2 1])
   b =
   Nonflat STREL object containing 3 neighbors.

   Neighborhood:
     1    1    1
   Height:
     1    2    1
```

将创建一个大小为 1 × 3 的结构元素，该元素的高度值为 $b(0, -1) = 1$，$b(0, 0) = 2$，$b(0, 1) = 1$。

与用于二值图像的方法相同，灰度图像的平坦的结构元素可使用函数 strel 来创建。例如，下面的命令显示了如何使用一个平坦的 3 × 3 结构元素来膨胀图 9.23(a)中的图像 f：

```
>> se = strel('square', 3);
>> gd = imdilate(f, se);
```

图 9.23(b)显示了结果。如同我们所预料的那样，图像稍微模糊了一些。下面我们讨论图中的其余部分。

结构元素 b 对 f 的灰度腐蚀记为 $f \ominus b$，定义为

$$(f \ominus b)(x, y) = \min\{f(x + x', y + y') - b(x', y') \mid (x', y') \in D_b\}$$

其中，D_b 为 b 的定义域，$f(x, y)$ 在 f 的定义域外假设为 $+\infty$。在概念上，我们可再次将该结构元素平移到图像中的所有位置。在每个平移后的位置，结构元素值会减去图像的像素值，因而所得的值是最小值。

与膨胀一样，灰度腐蚀通常使用平坦的结构元素来执行。平坦的灰度腐蚀公式可以简化为

$$(f \ominus b)(x, y) = \min\{f(x + x', y + y') \mid (x', y') \in D_b\}$$

因此，平坦的灰度腐蚀是一个局部最小值算子，其中的最小值取自由 D_b 的形状确定的一系列像素邻域。图 9.23(c)显示了使用函数 imerode 计算的结果，其中所用的结构元素与用于图 9.23(b)中的结构元素相同：

```
>> ge = imerode(f, se);
```

膨胀和腐蚀可以组合使用，以例获得各种效果。例如，从膨胀后的图像中减去腐蚀过的图像可产生一个"形态学梯度"，它是检测图像中局部灰度级变化的一种度量。例如，

```
>> morph_grad = imsubtract(gd, ge);
```

将产生如图 9.23(d)所示的图像，它是图 9.23(a)所示图像的"形态学梯度"。该图像具有边缘增长特征，这些特征类似于在 6.6.1 节和 10.1.3 节中讨论的梯度操作所得到的特征。

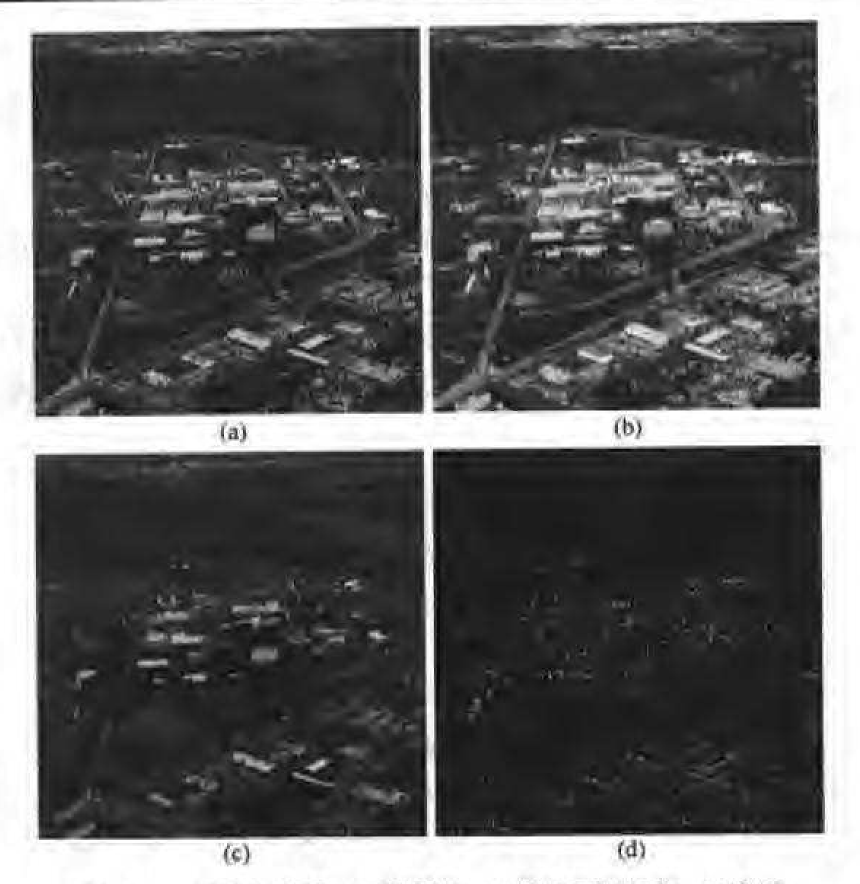


图 9.23 膨胀和腐蚀：(a)原图像；(b)膨胀后的图像；(c)腐蚀后的图像；(d)形态学梯度（原图像由 NASA 提供）

9.6.2 开运算和闭运算

灰度图像中开运算和闭运算的表达式与二值图像的相应表达式的形式相同。结构元素b对图像f的开运算记为$f \circ b$，定义为

$$f \circ b = (f \ominus b) \oplus b$$

与前面一样，这相当于f由b腐蚀，随后腐蚀的结果由b膨胀。类似地，b对f的闭运算定义为

$$f \bullet b = (f \oplus b) \ominus b$$

这两个运算都有简要的几何解释。假设一幅图像函数$f(x, y)$可看做是一个三维表面；换言之，其亮度值解释为xy平面上的高度值。然后，b对f的开运算可以解释为结构元素b沿表面f的下沿向上移动，并移过f的整个定义域。在结构元素沿f的底面滑动时，通过找到结构元素的任何部分所能到达的最高点，就可构建开运算。

图 9.24 示例了一维情形下的这种概念。图 9.24(a)所示的曲线是一幅图像中的单行取值。图9.24(b)在曲线下方显示了平坦的结构元素。完整的开运算在图9.24(c)中显示为沿着阴影区域顶部的曲线。由于结构元素太大而不能匹配曲线中间的峰值，所以开运算删除了该峰值。一般来说，开运算用来去除小的亮点，同时保持所有的灰度级和较大的亮区特性相对不变。

图 9.24(d)给出闭运算的图形示例。注意，结构元素位于曲线的上方，并沿曲线平移到了所有位置。图 9.24(e)显示了闭运算后的结果，其构建方式是当结构元素在曲线的上方滑动时，找到结构元素的任何部分所能到达的最低点。其中，我们看到闭运算去除了比结构元素更小的暗色细节。

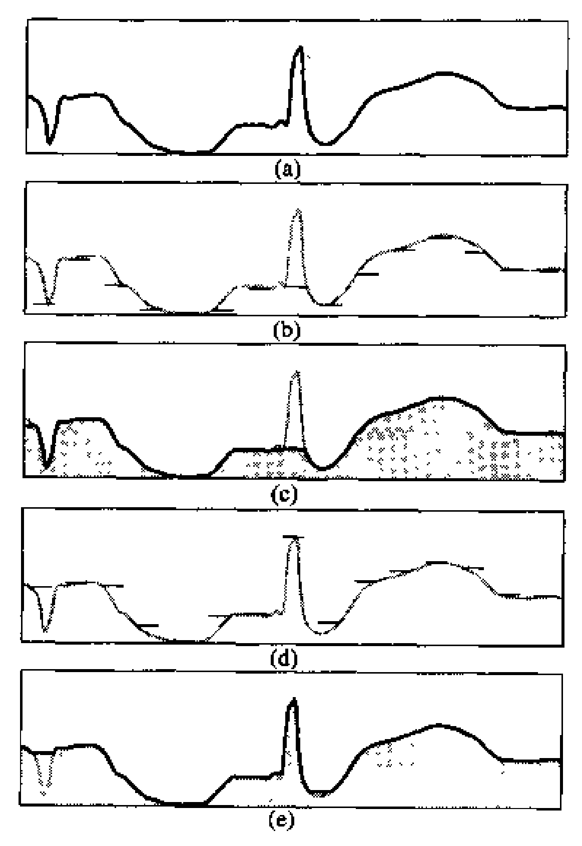


图 9.24　一维情形下的开运算和闭运算：(a)原始一维信号；(b)沿信号下方平移的的平坦的结构元素；(c)开运算；(d)沿信号的上方平移的平坦的结构元素；(e)闭运算

例 9.9　使用开运算和闭运算做形态学平滑

由于开运算可以去除比结构元素更小的明亮细节，闭运算可以去除比结构元素更小的暗色细节，所以它们经常组合在一起用来平滑图像并去除噪声。在本例中，我们使用函数 imopen 和 imclose 来平滑图 9.25(a)中所示的木接钉图片：

```
>> f = imread('plugs.jpg');
>> se = strel('disk', 5);
>> fo = imopen(f, se);
>> foc = imclose(fo, se);
```

图 9.25(b)显示了经开运算后的图像 fo，图 9.25(c)显示了经开运算再经闭运算后的图像 foc。我们可以看到背景和细节的平滑过程。这种过程通常称为开 - 闭滤波。闭 - 开滤波可以产生类似的结果。

另一种组合使用开运算和闭运算的方法是交替顺序滤波。交替顺序滤波的一种形式是用一系列不断增大的结构元素来执行开 - 闭滤波。下面的命令示例了该过程，开始时使用的是一个较小的结构元素，然后增加其大小，直到其大小与获得图 9.25(b)和图 9.25(c)所用的结构元素的大小相同为止：

```
>> fasf = f;
>> for k = 2:5
      se = strel('disk' ,k);
      fasf = imclose(imopen(fasf, se), se);
   end
```

在额外处理为代价的情形下，与单个开-闭滤波器相比，图9.25(d)所示的结果要稍微平滑一些。

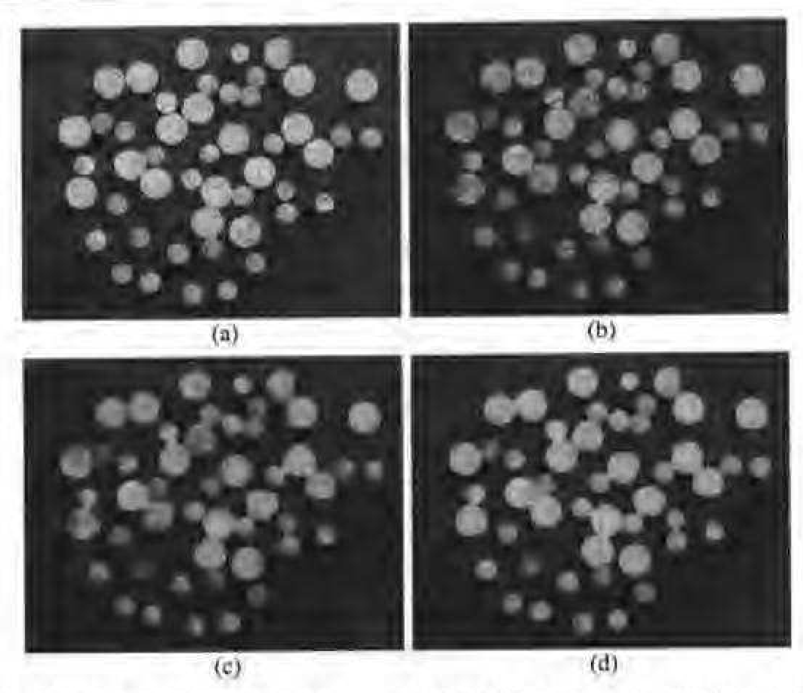


图 9.25　使用开运算和闭运算进行平滑：(a)木按钉的原图像；(b)使用半径为5的圆盘执行开运算后的图像；(c)经开运算再经闭运算后的图像；(d)交替顺序滤波后的图像

例 9.10　使用顶帽变换

开运算可用于补偿不均匀的背景亮度。图9.26(a)显示了一幅米粒图像 f，图像下部的背景要比上部的背景黑。对这样不均匀的亮度做阈值处理很困难（见10.3节）。例如，图9.26(b)是经阈值处理后的图像，图像顶部的米粒已被很好地从背景中分离开来，但图像底部的米粒并未从背景中正确地提取出来。对图像进行开运算可以产生对整个图像背景的合理估计，只要结构元素大到不能完全匹配米粒即可。例如，

```
>> se = strel('disk', 10);
>> fo = imopen(f, se);
```

产生了图9.26(c)所示的经开运算后的图像。从原图像中减去该图像，可以生成一幅具有合适且均匀的背景的米粒图像：

```
>> f2 = imsubtract(f, fo);
```

图9.26(d)显示了结果，图9.26(e)显示了经新的阈值处理后图像。可以看出，改进是很明显的。从原图像中减去开运算后的图像称为顶帽（top-hat）变换。IPT 函数 imtophat 只用一步就可执行该操作：

```
>> f2 = imtophat(f, se);
```

函数 imtophat 也可以像 g = imtophat(f,NHOOD) 这样来调用，其中 NHOOD 是一个值为 0 和 1 的数组，用于指定结构元素的大小和形状。该语法与使用调用 imtophat(f,strel(NHOOD)) 是一样的。

相关的函数 imbothat 可执行底帽（bottom-hat）变换，这种变换定义图像减去经闭运算后的图像。它的语法和 imtophat 的语法一样。这两个函数可以一起用于增强对比度，所用命令如下所示：

```
>> se = strel('disk', 3);
>> g = imsubtract(imadd(f, imtophat(f, se)), imbothat(f , se));
```

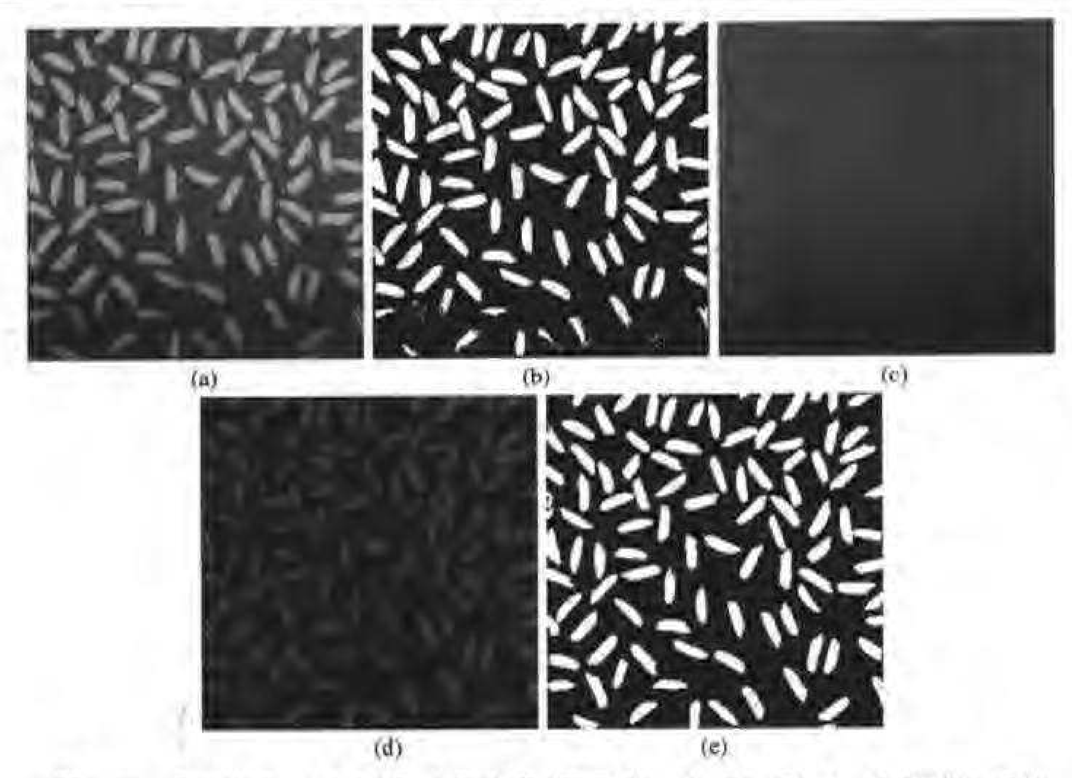


图9.26　顶帽变换：(a)原图像；(b)经阈值处理后的图像；(c)经开运算后的图像；(d)顶帽变换；(e)经阈值处理后的顶帽变换图像（原图像由MathWorks公司提供）

例9.11　颗粒分析

确定一幅图像中的颗粒大小分布的技术是颗粒分析技术领域的重要部分。形态学技术可以用于间接地度量颗粒的大小分布；换言之，不能准确地识别并度量每一个颗粒。对于形状规则且亮于背景的颗粒，基本方法是应用不断增大尺寸的形态学开运算。对于每一个开运算，开运算中的所有像素值的和会被计算；该和有时称为图像的表面积。下面的命令是对图9.25(a)所示图像进行半径为0到35的圆盘形开运算：

```
>> f = imread('plugs.jpg');
>> sumpixels = zeros(1, 36);
>> for k = 0:35
      se = strel('disk', k);
      fo = imopen(f, se);
      sumpixels(k + 1) = sum(fo(:));
   end
>> plot(0:35, sumpixels), xlabel('k'), ylabel('Surface area')
```

图9.27(a)显示了 sumpixels 与 k 的关系曲线。更为有趣的是，连续开运算之间的表面积会减小。

```
>> plot(-diff(sumpixels))
>> xlabel('k')
>> ylabel('Surface area reduction')
```

在图9.27(b)所示曲线的峰值表明出现了大量的有着这种半径的对象。因为曲线有很多的噪声，所以我们对9.25(d)所示的平滑后的图像重复这一过程。图9.27(c)所示的结果更清晰地指出了原图像中的两个不同尺寸的对象。

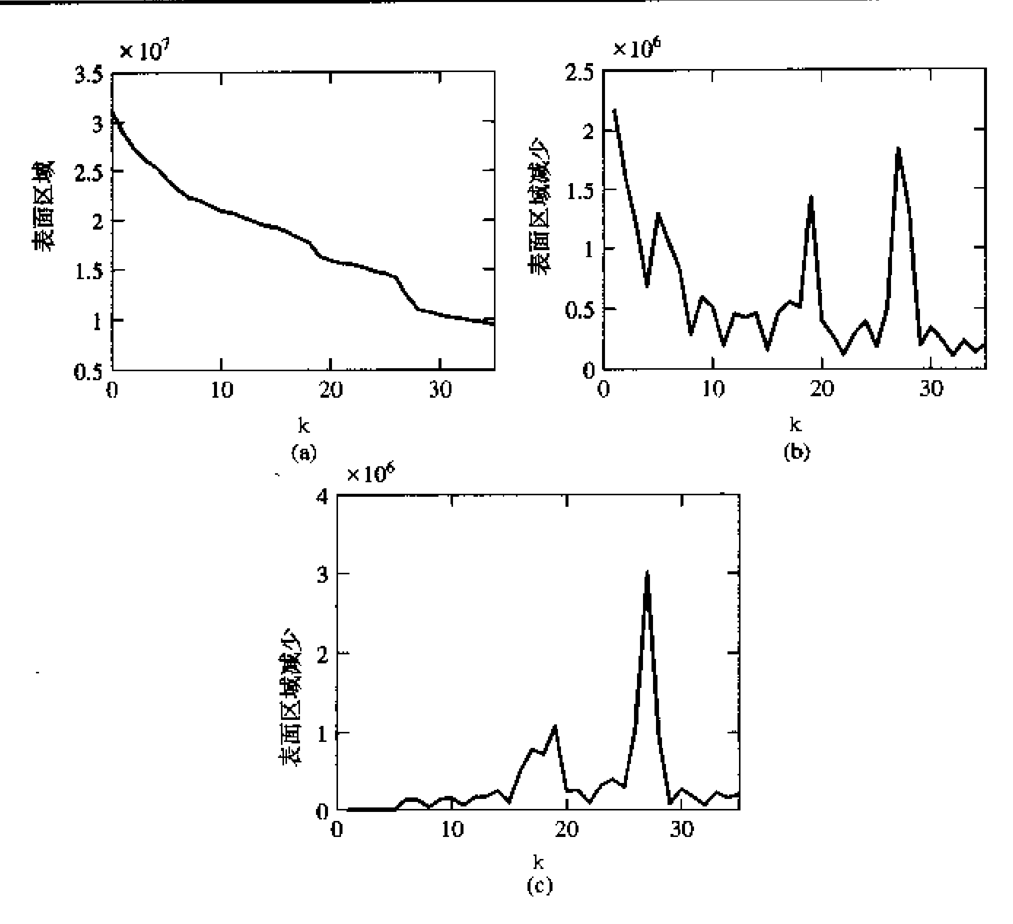


图 9.27　颗粒分析：(a)表面积与结构元素半径的关系；(b)减小的表面积与半径的关系；(c)平滑后的图像中，减小的表面积与半径的关系

9.6.3 重构

灰度级形态学重构由 9.5 节中给出的相同迭代过程定来定义。图 9.28 显示了一维情形下重构的工作方式。图 9.28(a)中的顶部曲线是掩模，底部的灰色曲线则是标记。这种情况下，标记由掩模减去一个常量形成。但是，通常情况下，任何信号都可以用做标记，只要不超过掩模板中的相应值即可。重构过程的每一次迭代都扩展了标记曲线的峰值，直到被掩模曲线强制压下为止[见图9.28(b)]。

最终的重构是图9.28(c)中的黑线。注意，重构中两个较小的峰值被消除了，但高一些的两个峰虽然被削弱了一些，但还是得到了保留。当一幅标记图像是由一幅掩模图像减去一个常量 h 得到的时，该重构就称为 h 极小值（h-minima）变换。由 IPT 函数 `imhmin` 计算的 h 极小值变换可用于抑制小峰值。

另一个有用的灰度级重构技术是开运算重构（opening-by-reconstruction），在这种重构中，一幅图像首先被腐蚀，就像在标准的形态学开运算中一样。但是，若使用闭运算代替开运算，则这幅被腐蚀的图像在图像重构中将用做标记图像。原图像将用做掩模。图9.29(a)显示了开运算重构的一个例子，它是使用如下命令实现的：

```
>> f = imread('plugs.jpg');
>> se = strel('disk', 5);
>> fe = imerode(f, se);
>> fobr = imreconstruct(fe, f);
```

通过应用一种称为闭运算重构（closing-by-reconstruction）的技术，可进一步清理图像 `fobr`。闭运算重构是通过对一幅图像求补、计算其开运算重构并对结果求补来实现的。步骤如下所示：

图9.30(e)显示了标准的顶帽计算的结果（即f_th）。

接下来，我们来消除图9.30(d)中键右侧的垂直反射光。通过使用一条短水平线来执行开运算重构，可以实现这一目的：

```
>> g_obr = imreconstruct(imerode(f_thr, ones(1, 11)), f_thr);
```

在结果中［见图9.30(f)］，垂直反射光已消失，但字符%中的斜线以及SIN中的"I"也消失了。我们利用了这样一个事实，即那些被误消了的字符非常接近于执行第一次膨胀后还存在的其他字符［见图9.30(g)］：

```
>> g_obrd = imdilate(g_obr, ones(1, 21));
```

接下来，通过将f_thr作为掩模，将min(g_obrd, f_thr)作为标记，我们进行最后的重构：

```
>> f2 = imreconstruct(min(g_obrd, f_thr), f_thr);
```

图9.30(h)显示了最后的结果。注意，背景和按键上阴影和反射光均已成功地去除了。

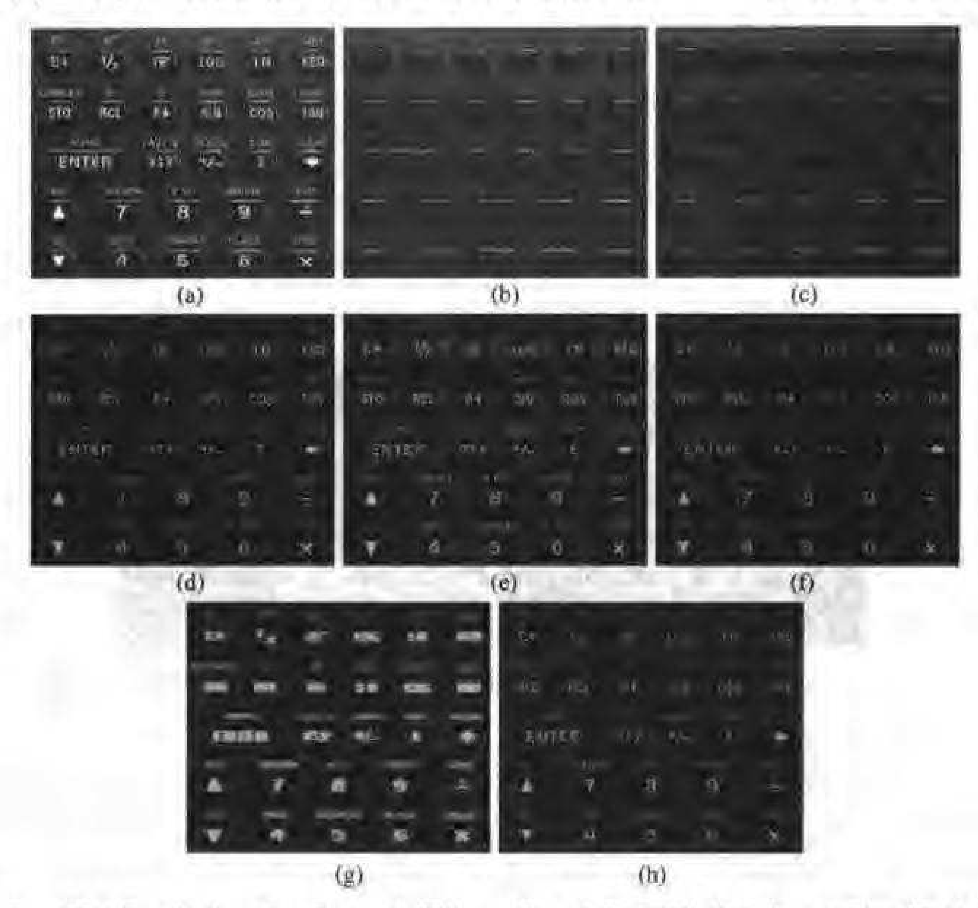


图9.30　灰度级重构的一个应用：(a)原图像；(b)经开运算重构后的图像；(c)开运算后的图像；(d)经顶帽重构后的图像；(e)经顶帽变换后的图像；(f)对图像(d)使用一条水平线开运算重构后的图像；(g)使用一条水平线对图像(f)膨胀后的图像；(h)最后的重构结果

小结

本章介绍的形态学概念和技术构成了一组提取图像特征的有力工具。针对二值图像和灰度图像的腐蚀、膨胀和重构的基本操作可以组合使用，以执行非常宽泛的任务。正像下一章将提到的那样，形态学技术也可用于图像分割。另外，如我们将在第11章讨论的那样，它们在图像描述的算法中也扮演着重要角色。

第10章　图像分割

前言

在第9章前，我们介绍的图像处理方法均是输入输出均为图像的情形；而从第9章起，我们便开始介绍输入是图像、输出是图像中提取出来的属性的图像处理方法。分割是该方向的另一个重要步骤。

分割将一幅图像细分为其组成区域或对象。细分的程度取决于要解决的问题。换言之，在应用中当感兴趣的对象已经被分割出来之后就应该停止分割。例如，电子装配线的自动检测所关注的是分析产品的图像，以确定对象是否存在特殊的异常，如元件缺失或连线断开等。因此，我们没有必要使用超过识别这些元件所需要的细节水平的分割。

非平凡图像的分割是图像处理中最困难的任务之一。分割的精度决定了计算机化的分析最终能否成功。因此，应该仔细考虑稳定分割的可能性。有些情况下（如工业检测方面的应用），至少在一定范围内控制某些测量是有可能的。其他情况下，例如在遥感中，用户对获取图像的控制主要受所选感器的限制。

单色图像的分割算法通常基于图像亮度值的两个基本特性：不连续性和相似性。在第一种类别中，处理方法是基于亮度的突变来分割一幅图像，如图像中的边缘。在第二种类别中，主要方法是根据事先定义的准则把图像分割成相似的区域。

在这一章中，我们将介绍刚才提到的两类方法，它们主要应用于单色图像。边缘检测和彩色图像分割将在6.6节中讨论。我们从开发适合于检测亮度的不连续性（如点、线、边缘等）的方法开始。边缘检测是分割算法的主要成果。除边缘检测外，我们还将使用基于Hough变换的方法来检测线性边缘分割。对边缘检测的讨论紧接在阈值技术之后。阈值技术也是分割技术中占有重要地位的基础处理方法，特别是速度为重要因素的应用场合。接下来将介绍面向区域分割方法的开发。我们将通过对名为分水岭分割的形态学分割方法的讨论来结束这一章。这种方法特别有吸引力，因为它将以全局形式产生一个明确且闭合的区域，并能提供一个框架，在该框架中，特别应用中关于图像的先验知识可以用来改善分割结果。

10.1　点、线和边缘检测

在本节中，我们将讨论在数字图像中检测亮度不连续的三种基本类型：点、线和边缘。用于查找不连续的最常用方法是在3.4节和3.5节中描述的方法，即对整幅图像运行一人掩模。对于大小为3×3的掩模来说，该过程将计算系数和由掩模覆盖区域所包含的灰度级的乘积之和。换言之，掩模在该图像中任何一点处的响应R由下式给出：

$$R = w_1z_1 + w_2z_2 + \cdots + w_9z_9 = \sum_{i=1}^{9} w_iz_i$$

其中，z_i是与掩模系数w_i相关的像素的亮度。和前面一样，掩模的响应的定义与其中心相关。

10.1.1　点检测

嵌在常数区域（或图像中亮度基本不变的区域）中的孤立点的检测，在原理上相当直接。在使用图 10.1 所示的掩模的情形下，若

$$|R| \geq T$$

则我们就可说在掩模的中心位置已检测出了一个孤立的点。其中，T 是一个非负阈值。点检测在 MATLAB 中可用函数 `imfilter` 来实现，所用的掩模与图 10.1 所示的掩模或其他掩模相似。最重要的需求是，当掩模的中心位于一个孤立点时，掩模的响应必须最强，而在亮度不变的区域中响应为零。

-1	-1	-1
-1	8	-1
-1	-1	-1

图 10.1　点检查模板

若 `T` 已给出，则如下命令可用实现刚才讨论的点检测方法：

```
>> g = abs(imfilter(double(f), w)) >= T;
```

其中，`f` 是输入图像，`w` 是一个合适的点检测掩模（如图 10.1 所示的掩模），`g` 是结果图像。回顾 3.4.1 节中的讨论可知，函数 `imfilter` 会将其输出转换为输入的类，若输入是 `uint8` 类，且 `abs` 操作不接受整数数据，则我们可在滤波操作中使用 `double(f)` 来防止过早的截断。若输出图像 `g` 是 `logical` 类图像，则它的值是 0 和 1。若未给出 `T` 值，则其值通常会基于滤波结果来选择，这种情况下，前面的命令串分成了三个基本步骤 (1)计算已滤波的图像，即 `abs(imfilter(double(f),w))`；(2)使用来自已滤波的图像的数据找到 `T` 的值；(3)将已滤波的图像与 `T` 做比较。这种方法将在下面的例子中说明。

例 10.1　点检测

图 10.2(a)显示了一幅图像，在该图像东北象限的暗灰色区域中有一个几乎看不见的黑点。若令 `f` 表示这幅图像，则我们可按如下方式找到该点的位置：

```
>> w = [-1 -1 -1; -1 8 -1; -1 -1 -1];
>> g = abs(imfilter(double(f), w));
>> T = max(g(:));
>> g = g >= T;
>> imshow(g)
```

通过将 `T` 选择为已滤波图像 `g` 中的最大值，然后在 `g` 中查找所有满足 `g >= T` 的点，我们就可识别这些有着最大响应的点。假设所有点都是嵌在不变或是基本不变的背景上的孤立点。注意，在对 `T` 的测试中，为保持符号的一致性，我们使用了 `>=` 操作符。本例中，`T` 已被选定为 `g` 中的最大值，很明显，`g` 中不可能有比 `T` 的值更大的点。如图 10.2(b)所示，其中有一个满足条件 `g >= T` 的孤立点，其中 `T` 置为 `max(g(:))`。

这种情况下，可以可以使用与该方向相关的掩模并对其输出做阈值处理，就像前一节中的等式那样。换言之，若我们对检测图像中由给定掩模定义的方向的所有线感兴趣，则可以简单地在图像上运行掩模并对结果的绝对值做阈值处理。剩下的点是响应最强烈的那些点，这些点与掩模定义的方向最为接近，且组成了只有一个像素宽的线。下面的例子说明了这个过程。

例 10.2 检测指定方向的线

图 10.4(a)显示了一个电路的数字化连线掩码图像（二值图像）。图像的大小为 486 × 486 像素。假设我们希望找到所有宽度为一个像素的线，且其方向为 −45°。为此目的，我们可使用图 10.3 中的最后一个掩模。图 10.4(b)到图 10.4(f)是使用下面的命令产生的，其中 f 是图 10.4(a)所示的图像：

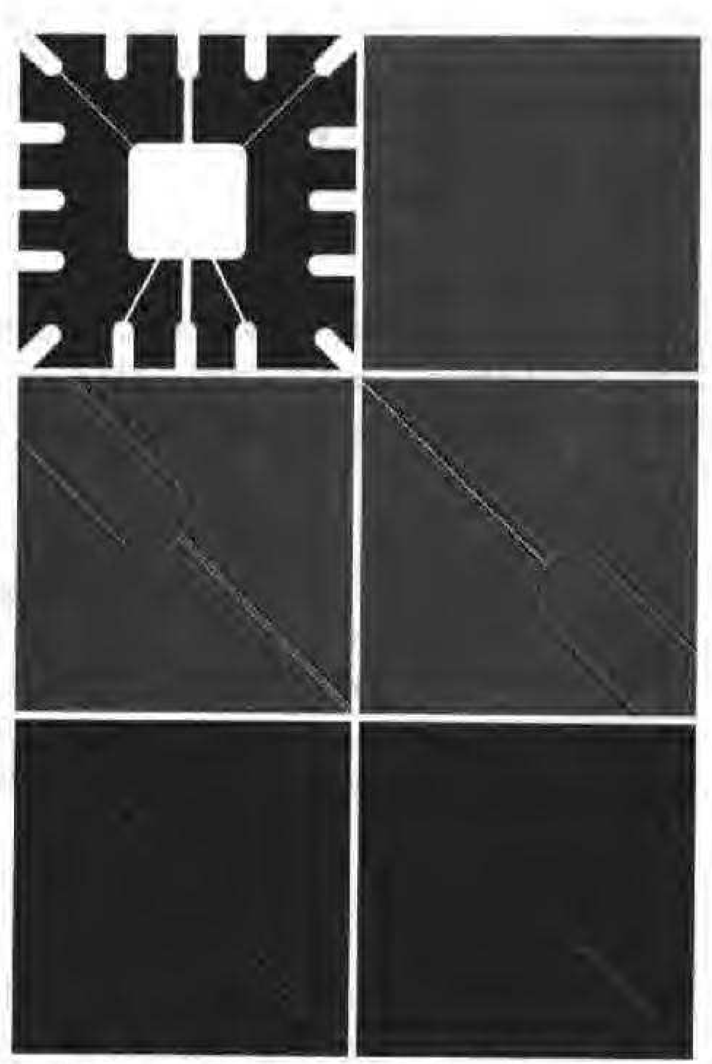


图 10.4 (a)连线掩模的图像；(b)使用图 10.3 中的 −45° 检测器处理后的结果；(c)图(b)的左上角放大图；(d)图(b)的右下角放大图；(e)图(b)的绝对值；(f)其值满足条件 g >= T 的所有点（白色），其中 g 是图(e)所示的图像［为便于查看，图(f)中的点被稍微放大］

```
>> w = [2 -1 -1 ; -1 2 -1; -1 -1 2];
>> g = imfilter(double(f), w);
>> imshow(g, [ ])  % Fig. 10.4(b)
>> gtop = g(1:120, 1:120);
>> gtop = pixeldup(gtop, 4);
>> figure, imshow(gtop, [ ]) % Fig. 10.4(c)
```

```
>> gbot = g(end-119:end, end-119:end);
>> gbot = pixeldup(gbot, 4);
>> figure, imshow(gbot, [ ]) % Fig. 10.4(d)
>> g = abs(g);
>> figure, imshow(g, [ ]) % Fig. 10.4(e)
>> T = max(g(:));
>> g = g >= T;
>> figure, imshow(g)  % Fig. 10.4(f)
```

暗于图10.4(b)中的灰色背景的阴影对应于负值。在$-45°$方向，存在两个主要的部分，一个在左上方,一个在右下方[图10.4(c)和图10.4(d)显示了这两个区域的放大片段]。注意，图10.4(d)中的直线部分要比图10.4(c)中的部分亮得多。原因是图10.4(a)中右下方的成分只有一个像素宽，左上方的成分则不是如此. 对于单个像素宽的成分，掩模响应更强。

图10.4(e)显示了图10.4(b)的绝对值。因为我们对最强的响应感兴趣，所以令T等于图像中的最大值。图10.4(f)显示了其值满足条件g>=T的白点，其中g是图10.4(e)所示的图像。图中的孤立点同样也是对掩模有着强烈响应的点。在原图像中,这些点及其紧邻点在那些孤立的位置处由掩模产生的最大响应来定向。这些孤立点可以通过图10.1中所示的掩模来检测和删除,或使用前一章讨论的形态学算子来检测。

10.1.3 使用edge函数的边缘检测

虽然点检测和线检测在任何关于图像分割的讨论中确实很重要,但到目前为止边缘检测最通用的方法是检测亮度值的不连续性。这样的不连续是用一阶和二阶导数检测的。图像处理中选择的一阶导数是在6.6.1节中定义过的梯度。为方便起见，我们在这里将重写相关的公式。二维函数$f(x, y)$的梯度定义为向量

$$\nabla\mathbf{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{bmatrix}$$

该向量的幅值是

$$\begin{aligned}\nabla f &= \mathrm{mag}(\nabla\mathbf{f}) = \left[G_x^2 + G_y^2\right]^{1/2} \\ &= \left[(\partial f/\partial x)^2 + (\partial f/\partial y)^2\right]^{1/2}\end{aligned}$$

为简化计算、该数值有时通过省略掉平方根的计算来近似，即

$$\nabla f \approx G_x^2 + G_y^2$$

或通过取绝对值来近似，即

$$\nabla f \approx |G_x| + |G_y|$$

这些近似值仍然具有导数性质；换言之，它们在不变亮度区中的值为零，而且它们的值与像素值可变区域中的亮度变化的程度成比例。在实际中，通常将梯度的幅值或它的近似值称为“梯度”。

梯度向量的基本性质是它指向f在(x, y)处的最大变化率方向。最大变化率出现时的角度为

$$\alpha(x, y) = \arctan\left(\frac{G_y}{G_x}\right)$$

关键问题之一是如何数字化地估计导数G_x和G_y。函数edge使用的各种方法将在本节后面讨论。

在图像处理中，二阶导数通常用 3.5.1 节中介绍的拉普拉斯算子来计算。换言之，二维函数 $f(x,y)$的拉普拉斯由二阶导数形成，如下式所示：

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$$

拉普拉斯算子自身很少被直接用做边缘检测，因为二阶导数对噪声具有无法接受的敏感性，它的幅度会产生双边缘，而且它不能检测边缘的方向。然而，正像本节稍后讨论的那样，当与其他边缘检测技术组合使用时，拉普拉斯算子是一种有效的补充方法。例如，虽然它的双边缘使得它不适合直接用于边缘检测，但该性质可用于边缘定位。

以前面的讨论为背景，边缘检测的基本意图是使用如下两个基本准则之一在图像中找到亮度快速变化的地方：

1. 找到亮度的一阶导数在幅度上比指定的阈值大的地方。
2. 找到亮度的二阶导数有零交叉的地方。

IPT 函数 edge 基于上面讨论的准则提供了几个导数估计器。对一些估计器来说，是有可能指定边缘检测器是否对水平和/或垂直边缘敏感的。该函数的基本语法为

```
[g, t] = edge(f, 'method', parameters)
```

其中，f 是输入图像，method 是表 10.1 中列出一种方法，parameters 是我们后面将要说明的另一个参数。在输出中，g 一个逻辑数组，其值如下决定：在 f 中检测到边缘的位置为 1，在其他位置为 0。参数 t 是可选的；它给出 edge 使用的阈值，以确定哪个梯度值足够大到可以称为边缘点。

表 10.1　函数 edge 中可用的边缘检测器

边缘检测器	基本特性
Sobel	使用图 10.5(b)所示的 Sobel 近似导数查找边缘
Prewitt	使用图 10.5(c)所示的 Prewitt 近似导数查找边缘
Roberts	使用图 10.5(d)所示的 Roberts 近似导数查找边缘
Laplacian of a Gaussian(LoG)	在使用高斯滤波器对 $f(x,y)$滤波之后，通过寻找零交叉来查找边缘
Zero crossings（零交叉）	使用一个自定义的滤波器对 $f(x,y)$滤波之后，通过寻找零交叉来查找边缘
Canny	通过寻找 $f(x,y)$的梯度的最大值来查找边缘。梯度由高斯滤波器的导数来计算。该方法使用两个阈值检测强边缘和弱边缘，若它们连结到了强边缘，则输出中只包含弱边缘。因此，这种方法更适合用于检测真正的弱边缘

Sobel 边缘检测器

Sobel 边缘检测器使用图 10.5(b)中的掩模来数字化地近似一阶导数值 G_x 和 G_y。换言之，一个邻域的中心点处的梯度可由 Sobel 检测器按如下方式计算：

$$\begin{aligned} g &= [G_x^2 + G_y^2]^{1/2} \\ &= \{[(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)]^2 \\ &\quad + [(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)]^2\}^{1/2} \end{aligned}$$

因此，我们说若在位置(x,y)处 $g \geq T$，则在该位置的一个像素是一个边缘像素，其中 T 是一个指定的阈值。

由 3.5.1 节的讨论我们可知，Sobel 边缘检测的实现过程如下：使用图 10.5(b)中左边的掩模对图像 f 进行滤波（使用 imfilter 函数），再使用另一个掩模对 f 滤波，然后计算每个滤波后的图像中的像素值的平方，并将两幅图像的结果相加，最后计算相加结果的平方根。类似的过程同样适用

于表10.1中的Prewitt和Roberts。函数edge只是将前面的操作封装到一个函数调用中，并添加一些其他的功能，如接受一个阈值或自动地确定一个阈值。此外，函数edge包含有不能直接使用imfilter实现的边缘检测技术。

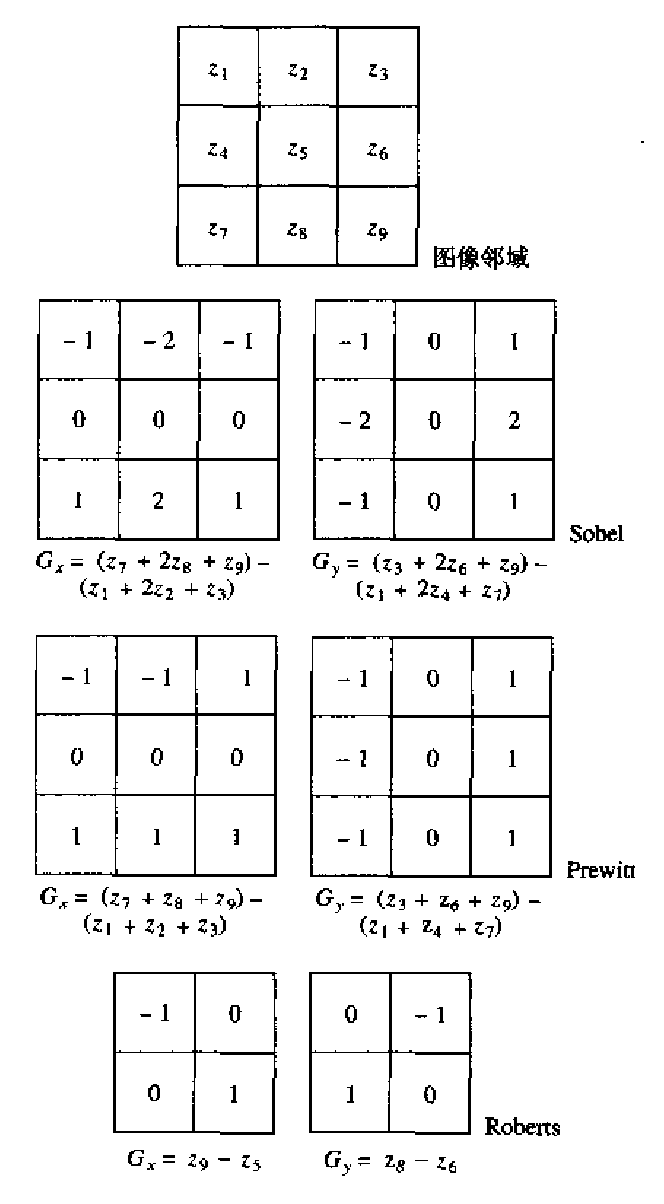


图10.5 一些边缘检测器掩模及其实现的一阶导数

Sobel检测器的调用语法为

```
[g, t] = edge(f, 'sobel', T, dir)
```

其中，f是输入图像，T是一个指定的阈值，dir指定检测边缘的首选方向：'horizontal'、'vertical'或'both'(默认值)。如先前说明的那样，g是在被检测到边缘的位置处为1而在其他位置为0的逻辑类图像。输出参数t是可选的，它是函数edge所用的阈值。若指定了T的值，则t = T。否则，若T未被赋值(或为空,[ ])，则函数edge会令t等于它自动确定的一个阈值，然后用于边缘检测。在输出参量中要包括t的主要原因之一是为了得到一个阈值的初始值。若使用语法g = edge(f)或[g, t] = egde(f)，则函数edge会默认使用Sobel检测器。

Prewitt 边缘检测器

Prewitt 边缘检测器使用图 10.5(c)中的掩模来数字化地近似一阶导数 G_x 和 G_y。其基本调用语法为

```
[g, t] = edge(f, 'prewitt', T, dir)
```

该函数的参数和 Sobel 的参数相同。Prewitt 检测器比 Sobel 检测器在计算上要简单一些，但比较容易产生一些噪声（系数为 2 的 Sobel 检测器可提供一种平滑作用）。

Roberts 边缘检测器

Roberts 边缘检测器使用图 10.5(d)中的掩模来数字化地近似一阶导数 G_x 和 G_y。其基本调用语法为

```
[g, t] = edge(f, 'roberts' , T, dir )
```

该函数的参数和 Sobel 的参数相同。Roberts 检测器是数字图像处理中最古老的边缘检测器之一，如图 10.5(d)所示，它也是最简单的一种边缘检测器。因为它具有一些功能上的限制，所以这种检测器的使用明显少于其他几种（例如，它是非对称的，而且不能检测诸如 45° 倍数的边缘）。然而，它还是经常用于硬件实现中，因为它既简单又快速。

Laplacian of a Gaussian（LoG）检测器

考虑高斯函数

$$h(r) = -e^{-\frac{r^2}{2\sigma^2}}$$

其中，$r^2 = x^2 + y^2$，σ 是标准偏差。这是一个平滑函数，若和一幅图像卷积，则会使图像变模糊。模糊的程度由 σ 的值决定。该函数的拉普拉斯算子（关于 r 的二阶导数）为

$$\nabla^2 h(r) = -\left[\frac{r^2 - \sigma^2}{\sigma^4}\right] e^{-\frac{r^2}{2\sigma^2}}$$

该函数称为 Laplacian of a Gaussian（LoG）有一些显而易见的原因。因为求二阶导数是线性运算，所以用 $\nabla^2 h(r)$ 对图像进行卷积（滤波）与先用平滑函数对图像卷积再计算结果的拉普拉斯算子是一样的。这是 LoG 检测器的最关键的概念。我们知道用 $\nabla^2 h(r)$ 对图像卷积会产生两个效果：使图像变平滑（从而减少噪声）；计算拉普拉斯算子，以便产生双边缘图像。然后，定位边缘就是找到两个边缘之间的零交叉。

LoG 检测器的基本调用语法为

```
[g, t] = edge(f, 'log' , T, sigma )
```

其中，sigma 是标准偏差，其他参数的解释如前所示。sigma 的默认值为 2。同前面一样，函数 edge 忽略一切比 T 值小的边缘线条。若 T 值未给出或为空[ ]，则 edge 会自动地选择一个值。令 T 为 0 会产生闭合轮廓的边缘，这是 LoG 方法的一个常见特征。

零交叉检测器

这种检测器基于与 LoG 方法相同的概念，但卷积是使用指定的滤波函数 H 执行的。调用语法为

```
[g, t] = edge(f, 'zerocross', T, H )
```

其他参数的解释与 LoG 中的参数相同。

Canny 边缘检测器

Canny 检测器（Canny[1986]）是使用函数 edge 的最有效边缘检测器。该方法总结如下：

1. 图像使用带有指定标准偏差σ的高斯滤波器来平滑，从而可以减少噪声。
2. 在每一点处计算局部梯度$g(x, y) = [G_x^2 + G_y^2]^{1/2}$ 和边缘方向$\alpha(x, y) = \arctan(G_y/G_x)$。表10.1中的前三项都可以用来计算$G_x$和$G_y$。边缘点定义为梯度方向上其强度局部最大的点。
3. 第2条中确定的边缘点会导致梯度幅度图像中出现脊。然后，算法追踪所有脊的顶部，并将所有不在脊的顶部的像素设为零，以便在输出中给出一条细线，这就是众所周知的非最大值抑制处理。脊像素使用两个阈值 *T1* 和 *T2* 做阈值处理，其中 *T1* < *T2*。值大于 *T2* 的脊像素称为强边缘像素，*T1* 和 *T2* 之间的脊像素称为弱边缘像素。
4. 最后，算法通过将 8 连接的弱像素集成到强像素，执行边缘链接。

Canny 边缘检测器的语法为

```
[g, t] = edge(f, 'canny' , T, sigma )
```

其中，T 是一个向量，T = [T1,T2]，它包含在步骤3中已解释过的两个阈值；sigma 是平滑滤波器的标准偏差。若 t 包含在输出参量中，则它是一个二元向量，该向量包含有该算法用到的两个阈值。语法中的其余参量和其他方法中解释的一样，包括未指定 T 的情况下自动计算阈值。sigma 的默认值为 1。

例 10.3 使用 Sobel 检测器来提取边缘

我们可以使用如下命令来提取并显示图 10.6(a)所示图像 f 的垂直边缘：

```
>> [gv, t] = edge(f, 'sobel', 'vertical');
>> imshow(gv)
>> t

t =
    0.0516
```

如图 10.6(b)所示，结果中的主要边缘是垂直边缘（倾斜的边缘有垂直分量和水平分量，所以也能被检测到）。我们可以指定一个较高的阈值，从而把弱一些的边缘去掉。例如，图 10.6(c)是用以下命令产生的：

```
>> gv = edge(f, 'sobel', 0.15, 'vertical');
```

在以下命令中使用相同的 T 值可产生图 10.6(d)所示的结果，它突出显示了垂直边缘和水平边缘：

```
>> gboth = edge(f, 'sobel', 0.15);
```

如图函数 edge 不能计算 ±45° 的 Sobel 边缘。要计算这些边缘，我们需要指定掩模并使用函数 imfilter。例如，图 10.6(e)是由如下命令产生的：

```
>> w45 = [-2 -1 0; -1 0 1; 0 1 2]

w45 =

    -2    -1     0
    -1     0     1
     0     1     2

>> g45 = imfilter(double(f), w45, 'replicate');
```

```
>> T = 0.3*max(abs(g45(:)));
>> g45 = g45 >= T;
>> figure, imshow(g45);
```

图10.6(e)中的最强边缘是方向为45°的边缘。类似地，使用掩模 45 =[0 1 2;-1 0 1; -2 -1 0] 以及相同的命令序列，我们可以得到方向为-45°的强边缘，如10.6(f)所示。

在函数edge中使用选项'prewitt'和'roberts'的过程，类似于使用Sobel边缘检测器的过程。

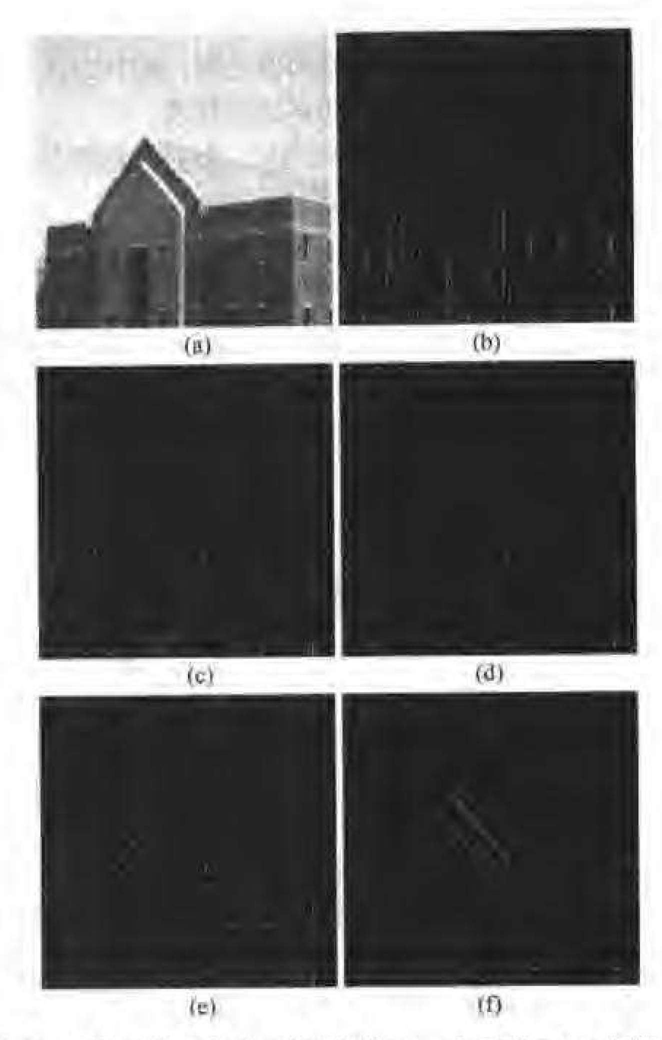


图10.6 (a)原图像；(b)使用带有自动确定的阈值的一个垂直Soble掩模后，函数edge导致的结果；(c)使用指定阈值后的结果；(d)使用指定阈值来决定垂直边缘和水平边缘的结果；(e)使用函数imfilter（具有指定的掩模和阈值）计算45°边缘的结果；(f)使用函数imfilter（具有指定的掩模和阈值）计算-45°边缘的结果

例10.4 Sobel，LoG和Canny边缘检测器的比较

在这个例子中，我们将比较Sobel，LoG和Canny边缘检测器的相关性能。我们的目的是，通过提取图10.6(a)所示的房屋图像f的主要边缘特征并去掉不重要的细节（如砖墙和瓦片屋顶的纹理），从而产生一个干净的边缘"映射"。在讨论中，我们感兴趣的主要边缘是房屋的角落、窗户、形成入口的亮砖结构和入口本身、屋顶以及房屋高度的三分之二处的水泥条带。

图10.7左边的一列显示了使用默认的选项'sobel'、'log'和'canny'得到的边缘图像：

```
>> [g_sobel_default, ts] = edge(f, 'sobel'); % Fig. 10.7(a)
```

10.2 使用 Hough 变换的线检测

理想情况下，前一节讨论的方法应该只产生边缘上的像素。实际上，结果像素由于噪声、不均匀照明引起的边缘断裂和杂散的亮度不连续而难以得到完全的边缘特性。因此，典型的边缘检测算法遵循用链接过程把像素组装成有意义的边缘的方法。一种寻找并链接图像中线段的处理方式是 Hough 变换（Hough[1962]）。

给定一幅图像（一般为二值图像）中的一个点集合，假设我们想要找到位于直线上的所有点的子集。一种可能的解决方法就是先找到所有由每一对点决定的线，然后找到接近特殊线的所有点的子集。这种处理方法的问题是它需要找 $n(n-1)/2 \sim n^2$ 条线，然后进行 $n(n(n-1))/2 \sim n^3$ 次每一点与所有线的比较。这种方法计算起来很麻烦，因而一般不予采用。

另一方面，采用 Hough 变换时，我们考虑一个点(x_i, y_i)和所有通过该点的线。点(x_i, y_i)的线有无数条，这些线对某些 a 值和 b 值来说，均满足 $y_i = ax_i + b$。将该公式写为 $b = -x_i a + y_i$ 并考虑 ab 平面（也称为参数空间），可对一个固定点(x_i, y_i)产生单独的一条直线。此外，第二个点(x_j, y_j)也有这样一条在参数空间上与它相关的直线，这条直线和与(x_i, y_i)相关的直线相交于点(a', b')，其中 a' 和 b' 分别是 xy 平面上包含点(x_i, y_i)和(x_j, y_j)的直线的斜率和截距。事实上，在这条直线上的所有点都有在参数空间中相交于点(a', b')的直线。图 10.8 说明了这些概念。

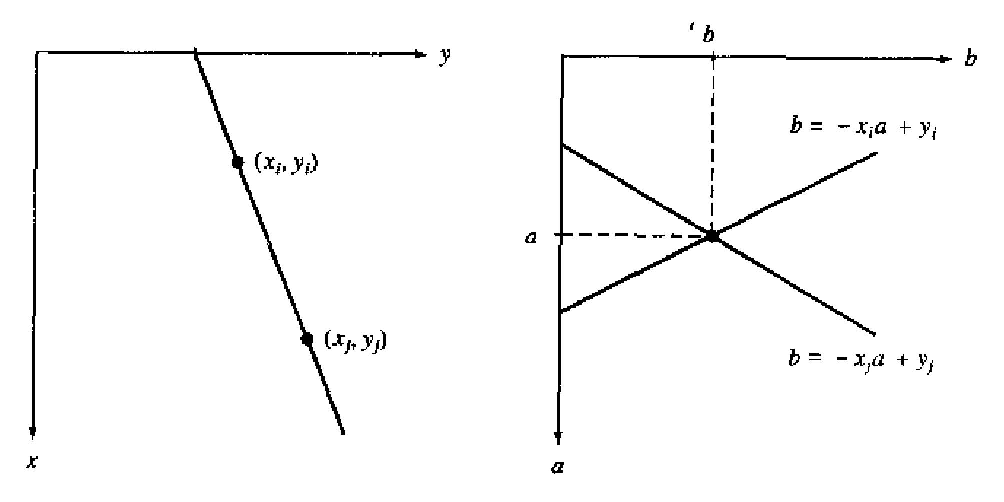


图 10.8 (a)xy 平面；(b)参数空间

从原理上讲，我们可以绘制出与所有图像点(x_i, y_i)相对应的参数 - 空间直线，而图像直线可以通过许多参数 - 空间直线相交来识别。然而，这种方法的实际困难是 a（直线的斜率）接近无限大，也就是接近垂直方向。解决该困难的一种方法是使用直线的标准表示法：

$$x\cos\theta + y\sin\theta = \rho$$

图 10.9(a)说明了参数 ρ 和 θ 的几何解释。对于水平线来说，$\theta = 0°$，ρ 等于正的 x 截距。同样，对于垂直线而言，$\theta = 90°$，ρ 等于正的 y 截距，或 $\theta = -90°$，ρ 等于负的 y 截距。图 10.9(b)中的每一条正弦曲线表示通过特定点(x_i, y_i)的一族直线。交点(ρ', θ')对应于通过(x_i, y_i)和(x_j, y_j)的直线。

Hough 变换计算上的诱人之处是把 $\rho\theta$ 参数空间细分为了所谓的累加器单元，如图 10.9(c)所示，其中(ρ_{min}, ρ_{max})和$(\theta_{min}, \theta_{max})$是参数值的期望范围。一般来说，值的最大范围是 $-90° \leq \theta \leq 90°$ 和 $-D \leq \rho \leq D$，其中 D 是图像中角点间的距离。坐标(i, j)处的单元（累加器的值为 $A(i, j)$）对应于与参数空间坐标(ρ_i, θ_j)相关的方形。最初，这些单元被设为零。然后，对于图像平面上的每一个非背景点(x_k, y_k)，我们令 θ 等于 θ 轴上允许的细分值，并通过公式 $\rho = x_k\cos\theta + y_k\sin\theta$ 求出相应的 ρ 值。

然后，将得到的ρ值四舍五入为最接近的、ρ轴上的允许单元值。相应的累加器单元然后增加。在这个过程的最后，$A(i, j)$意味着 xy 平面上的 Q 个点位于线 $x\cos\theta_j + y\sin\theta_j = \rho_i$ 上。$\rho\theta$平面上的细分数决定了这些点共线的精度。

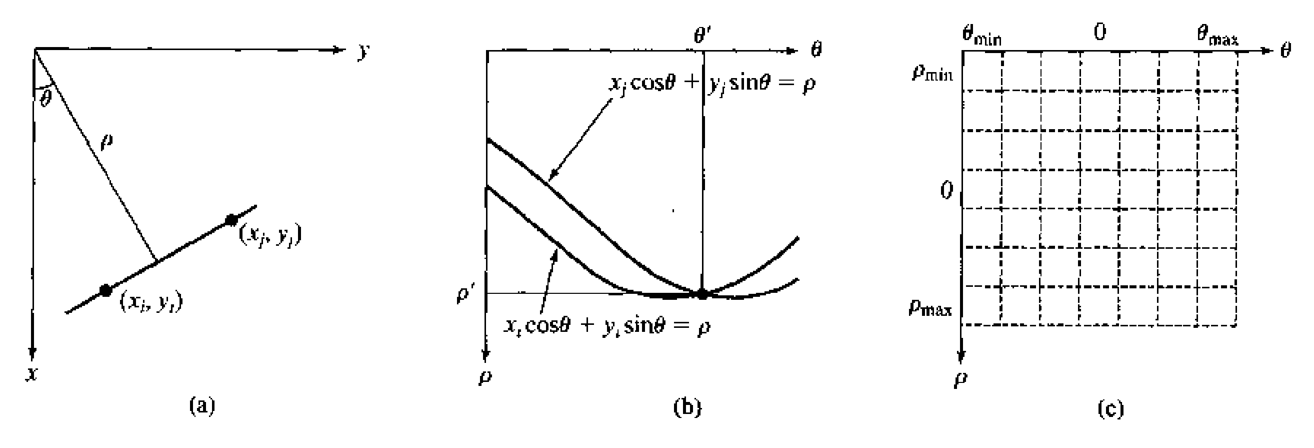


图 10.9 (a)在xy平面上的线的(ρ, θ)参数化；(b)$\rho\theta$平面上的正弦曲线，交点(ρ', θ')对应于连接(x_i, y_i)和(x_j, y_j)的线的参数；(c)把$\rho\theta$平面划分为累加器单元

计算Hough变换的函数在下面给出。该函数利用了稀疏矩阵，即其中只有很少非零元素的矩阵。这种特性提供了矩阵存储空间和计算时间的优点。给出一个矩阵`A`，我们使用函数`sparse`把它转换成稀疏矩阵，该函数的基本语法为

```
S = sparse(A)
```

例如，

```
>> A = [ 0   0   0   5
         0   2   0   0
         1   3   0   0
         0   0   4   0 ];
>> S = sparse(A)

S =
     (3,1)     1
     (2,2)     2
     (3,2)     3
     (4,3)     4
     (1,4)     5
```

该输出列出了`S`中的非零元素及其行和列索引。元素按列排序。

函数`sparse`的更常用的语法由如下5个参量组成：

```
S = sparse(r, c, s, m, n)
```

其中，`r`和`c`分别是我们希望转换为稀疏矩阵的矩阵中非零元素的行和列索引向量。参数`s`是一个向量，它包含有相应于索引对`(r,c)`的值，`m`和`n`是结果矩阵的行维数和列维数。例如，前一个例子中的矩阵`S`可以使用如下命令直接产生：

```
>> S = sparse([3 2 3 4 1], [1 2 2 3 4], [1 2 3 4 5], 4, 4);
```

函数`sparse`还有许多其他的语法形式，详见帮助页中对该函数的说明。

给出一个由任意可用的语法形式产生的稀疏矩阵`S`后，我们就可以使用函数`full`来获得完整的矩阵，函数`full`的语法为

```
    y_matrix     = repmat(y(first:last), 1, ntheta);
    val_matrix   = repmat(val(first:last), 1, ntheta);
    theta_matrix = repmat(theta, size(x_matrix, 1), 1)*pi/180;

    rho_matrix = x_matrix.*cos(theta_matrix) + ...
        y_matrix.*sin(theta_matrix);
slope = (nrho - 1)/(rho(end) - rho(1));
rho_bin_index = round(slope*(rho_matrix - rho(1)) + 1);

theta_bin_index = repmat(1:ntheta, size(x_matrix, 1), 1);

% Take advantage of the fact that the SPARSE function, which
% constructs a sparse matrix, accumulates values when input
% indices are repeated. That's the behavior we want for the
% Hough transform. We want the output to be a full (nonsparse)
% matrix, however, so we call function FULL on the output of
% SPARSE.
h = h + full(sparse(rho_bin_index(:), theta_bin_index(:), ...
                    val_matrix(:), nrho, ntheta));
end
```

例 10.5 Hough 变换的说明

在这个例子中，我们将用一幅简单的二值图像来说明hough函数的用法。首先，我们建立一幅在一些位置存在孤立前景像素的图像。

```
>> f = zeros(101, 101);
>> f(1, 1)     = 1;  f(101, 1) = 1; f(1, 101) = 1;
>> f(101, 101) = 1;  f(51, 51) = 1;
```

图10.10(a)显示了我们的测试图像。下面我们计算并显示Hough变换。

```
>> H = hough(f);
>> imshow(H, [ ])
```

图10.10(b)显示了结果，该结果是用函数imshow来显示的。但在带有标度轴的较大图形中显示结果将更利于我们查看Hough变换。在接下来的代码段中，我们使用三个参量调用了hough函数；第二个两输出参量包括分别对应于Hough变换矩阵的每一列和每一行的 θ 和 ρ 值。这些向量（theta 和rho）可以作为附加输入参量传递给imshow来控制水平轴和垂直轴的标度。我们也可以将选项'notruesize'传递给函数imshow。函数axis用来打开轴标记，并在显示结果中带有矩形。最后，函数xlabel和函数ylabel（见3.3.1节）用于将希腊字母标在轴上。

```
>> [H, theta, rho] = hough(f);
>> imshow(theta, rho, H, [ ], 'notruesize')
>> axis on, axis normal
>> xlabel('\theta'), ylabel('\rho')
```

图10.10(c)显示了标值后的结果。三条正弦曲线在±45°处的交点表明f中存在两组三个共线的点。两条正弦曲线在$(\theta, \rho) = (-90, 0)$、$(-90, -100)$、$(0, 0)$、$(0, 100)$处的交点表明存在四组位于垂直线和水平线上的共线点。

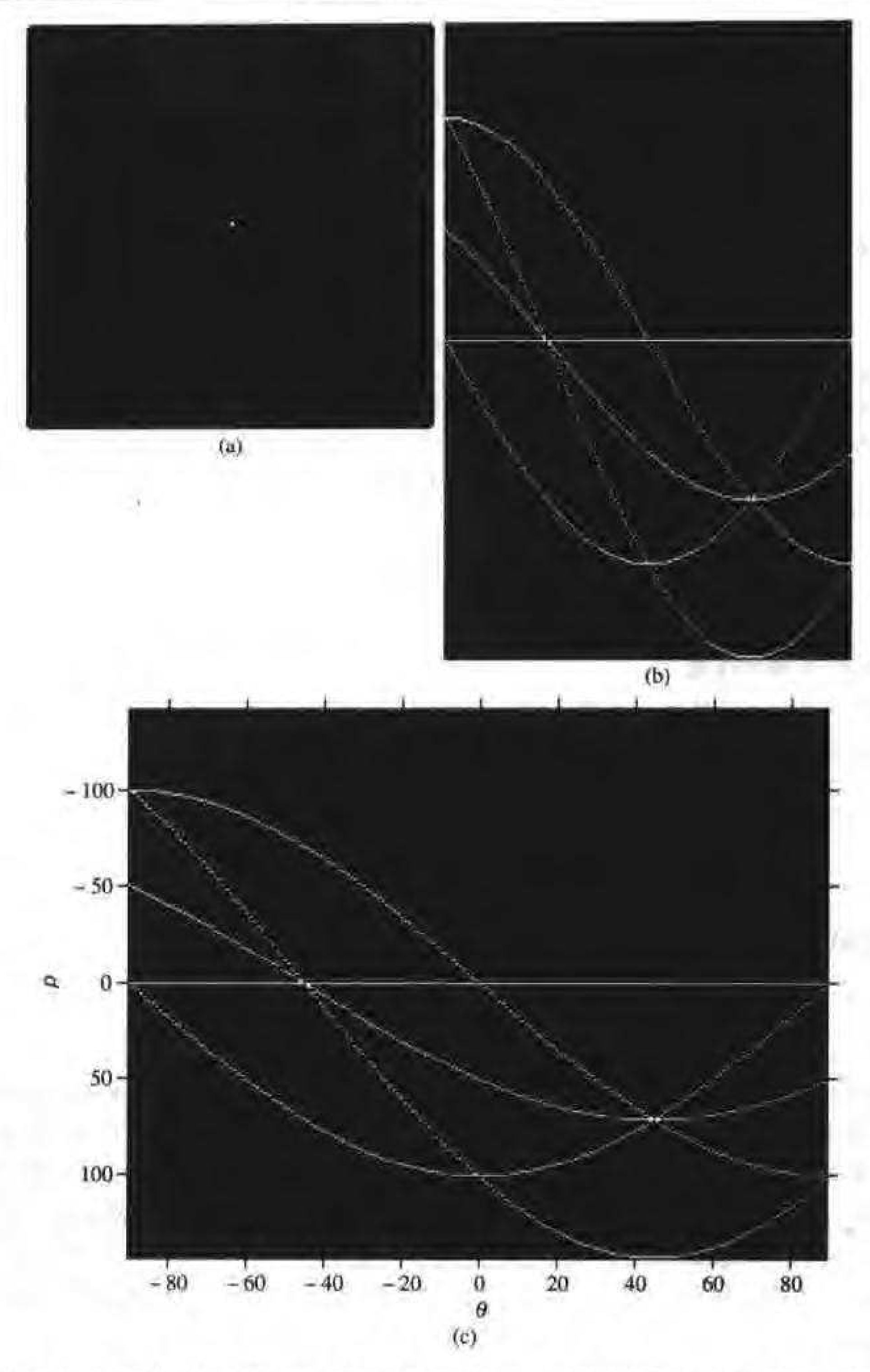


图 10.10　(a)带有5个点的二值图像（4个点在角上）；(b)使用函数 imshow 显示的 Hough 变换；(c)带有标度轴的 Hough 变换（为便于观看，图(a)中的点已被放大）

10.2.1　使用 Hough 变换做峰值检测

使用 Hough 变换进行线检测和链接的第一步是峰值检测。在 Hough 变换中找到一组有意义的明显峰值是一种挑战。因为数字图像的空间中的量化、Hough 变换的参数空间中的量化以及典型图

像的边缘都不是很直这个事实，Hough变换的峰值一般都位于多个Hough变换单元中。克服这个问题的一个策略如下所示：

1. 找到包含有最大值的Hough变换单元并记下它的位置。
2. 把第一步中找到的最大值点的邻域中的Hough变换单元设为零。
3. 重复该步骤，直到找到需要的峰值数时为止，或者达到一个指定的阈值时为止。

函数houghpeaks可用于实现该策略。

```
function [r, c, hnew] = houghpeaks(h, numpeaks, threshold, nhood)
%HOUGHPEAKS Detect peaks in Hough transform.
%   [R, C, HNEW] = HOUGHPEAKS(H, NUMPEAKS, THRESHOLD, NHOOD) detects
%   peaks in the Hough transform matrix H. NUMPEAKS specifies the
%   maximum number of peak locations to look for. Values of H below
%   THRESHOLD will not be considered to be peaks. NHOOD is a
%   two-element vector specifying the size of the suppression
%   neighborhood. This is the neighborhood around each peak that is
%   set to zero after the peak is identified. The elements of NHOOD
%   must be positive, odd integers. R and C are the row and column
%   coordinates of the identified peaks. HNEW is the Hough transform
%   with peak neighborhood suppressed.
%
%   If NHOOD is omitted, it defaults to the smallest odd values >=
%   size(H)/50. If THRESHOLD is omitted, it defaults to
%   0.5*max(H(:)). If NUMPEAKS is omitted, it defaults to 1.

if nargin < 4
   nhood = size(h)/50;
   % Make sure the neighborhood size is odd.
   nhood = max(2*ceil(nhood/2) + 1, 1);
end
if nargin < 3
   threshold = 0.5 * max(h(:));
end
if nargin < 2
   numpeaks = 1;
end

done = false;
hnew = h; r = []; c = [];
while ~done
   [p, q] = find(hnew == max(hnew(:)));
   p = p(1); q = q(1);
   if hnew(p, q) >= threshold
      r(end + 1) = p; c(end + 1) = q;

      % Suppress this maximum and its close neighbors.
      p1 = p - (nhood(1) - 1)/2; p2 = p + (nhood(1) - 1)/2;
      q1 = q - (nhood(2) - 1)/2; q2 = q + (nhood(2) - 1)/2;
      [pp, qq] = ndgrid(p1:p2, q1:q2);
      pp = pp(:); qq = qq(:);

      % Throw away neighbor coordinates that are out of bounds in
      % the rho direction.
      badrho = find((pp < 1) | (pp > size(h, 1)));
      pp(badrho) = []; qq(badrho) = [];
```

```
      % For coordinates that are out of bounds in the theta
      % direction, we want to consider that H is antisymmetric
      % along the rho axis for theta = +/- 90 degrees.
      theta_too_low = find(qq < 1);
      qq(theta_too_low) = size(h, 2) + qq(theta_too_low);
      pp(theta_too_low) = size(h, 1) - pp(theta_too_low) + 1;
      theta_too_high = find(qq > size(h, 2));
      qq(theta_too_high) = qq(theta_too_high) - size(h, 2);
      pp(theta_too_high) = size(h, 1) - pp(theta_too_high) + 1;

      % Convert to linear indices to zero out all the values.
      hnew(sub2ind(size(hnew), pp, qq)) = 0;
      done = length(r) == numpeaks;
   else
      done = true;
   end
end
```

函数 houghpeaks 将在例 10.6 中解释。

10.2.2 使用 Hough 变换做线检测和链接

一旦在 Hough 变换中识别出了一组候选的峰被，则它们还要留待确定是否存在与这些峰值相关的线段以及它们的起始和结束位置。对每一个峰值来说，第一步是找到图像中影响到峰值的每一个非零值点的位置。为达到这一目的，我们编写了函数 houghpixels。

```
function [r, c] = houghpixels(f, theta, rho, rbin, cbin)
%HOUGHPIXELS Compute image pixels belonging to Hough transform bin.
%   [R, C] = HOUGHPIXELS(F, THETA, RHO, RBIN, CBIN) computes the
%   row-column indices (R, C) for nonzero pixels in image F that map
%   to a particular Hough transform bin, (RBIN, CBIN). RBIN and CBIN
%   are scalars indicating the row-column bin location in the Hough
%   transform matrix returned by function HOUGH. THETA and RHO are
%   the second and third output arguments from the HOUGH function.

[x, y, val] = find(f);
x = x - 1; y = y - 1;

theta_c = theta(cbin) * pi / 180;
rho_xy = x*cos(theta_c) + y*sin(theta_c);
nrho = length(rho);
slope = (nrho - 1)/(rho(end) - rho(1));
rho_bin_index = round(slope*(rho_xy - rho(1)) + 1);

idx = find(rho_bin_index == rbin);

r = x(idx) + 1; c = y(idx) + 1;
```

与使用函数 houghpixels 找到的位置相关的像素必须组合成线段。函数 houghlines 采用下面的策略：

1. 将像素位置旋转 $90°-\theta$，以便它们大概位于一条垂直线上。
2. 按旋转的 x 值来对这些像素位置分排序。
3. 使用函数 diff 找到裂口。忽视掉小裂口；这将合并被小空白分离的相邻线段。
4. 返回比最小阈值长的线段的信息。

10.3 阈值处理

由于实现的直观性和简单性，图像阈值处理在图像分割应用中占有重要的地位。简单的阈值处理已在2.7.2节中介绍过，并且在前面章节的讨论中已经使用过。本节我们将讨论自动选择阈值的方法，此外还要介绍一种依照局部图像邻域的性质来改变阈值的方法。

假设图10.12所示的直方图对应于图像$f(x, y)$，该图像由暗以背景上的较亮对象组成，采用这种方法，目标和背景像素会具有两种主要模式的灰度级。一种从背景上提取对象的明显方法是选取一个阈值T来分离这两种模式。任何满足$f(x, y) \geqslant T$的点(x, y)称为对象点，其他点则称为背景点。换言之，阈值处理后的图像$g(x, y)$定义为

$$g(x, y) = \begin{cases} 1 & \text{如果} f(x, y) \geqslant T \\ 0 & \text{如果} f(x, y) < T \end{cases}$$

标注为1的像素对应于对象，而标注为0的像素则对应于背景。T为常数时，这种方法称为全局阈值处理。选取全局阈值的方法将在10.3.1节中讨论。在10.3.2节中，我们将讨论允许阈值变化的方法，该方法称为局部阈值处理。

10.3.1 全局阈值处理

选取阈值的一种方法是目视检查直方图。图10.12所示的直方图有两个不同的模式；正如结果所示，我们可很容易地选取一个阈值T来分开它们。另一种选择T的方法是反复试验，挑选不同的阈值，直到观测者觉得产生了较好的结果时为止。这在交互式环境下特别有效，例如，这种方法允许用户使用一个图形控件（如滑动条）来改变阈值并可立即看到结果。

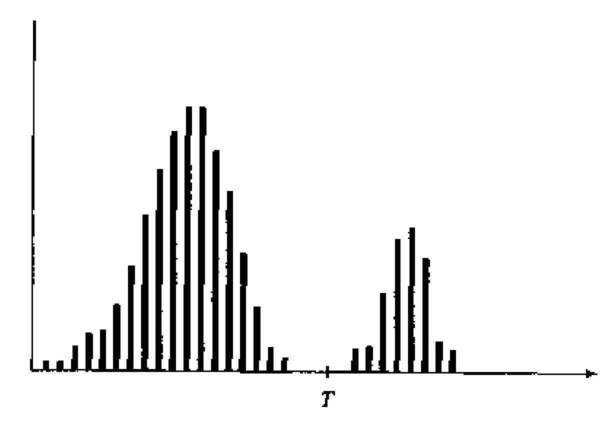


图10.12 通过目视分析一个双模式直方图来选择阈值

为自动选择一个阈值，Gonzalez和Woods[2002]描述了如下迭代步骤：

1. 为T选一个初始估计值（建议初始估计值为图像中最大亮度值和最小亮度值的中间值）。
2. 使用T分割图像。这会产生两组像素：亮度值$\geqslant T$的所有像素组成的G_1，亮度值$<T$的所有像素组成的G_2。
3. 计算G_1和G_2范围内的像素的平均亮度值μ_1和μ_2。
4. 计算一个新阈值：

$$T = \frac{1}{2}(\mu_1 + \mu_2)$$

5. 重复步骤2到步骤4，直到连续迭代中T的差比预先指定的参数T_o小为止。

我们将在例10.7中说明如何使用MATLAB来实现这些操作步骤。

工具箱提供了一个称为 graythresh 的函数，该函数使用Otsu方法（Otsu[1979]）来计算阈值。为检验这种基于直方图的方法，我们从处理一个离散概率密度函数的归一化直方图开始，如下所示：

$$p_r(r_q) = \frac{n_q}{n} \qquad q = 0, 1, 2, \cdots, L-1$$

其中，n是图像中的像素总数，n_q是灰度级为r_q的像素数目，L是图像中所有可能的灰度级数。假设我们现在已经选定了一个阈值k，C_0是一组灰度级为$[0, 1, \cdots, k-1]$的像素，C_1是一组灰度级为$[k, k+1, \cdots, L-1]$的像素。Otsu 方法选择最大化类间方差σ_B^2的阈值k，类间方差定义为

$$\sigma_B^2 = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2$$

其中，

$$\omega_0 = \sum_{q=0}^{k-1} p_q(r_q)$$

$$\omega_1 = \sum_{q=k}^{L-1} p_q(r_q)$$

$$\mu_0 = \sum_{q=0}^{k-1} q p_q(r_q)/\omega_0$$

$$\mu_1 = \sum_{q=k}^{L-1} q p_q(r_q)/\omega_1$$

$$\mu_T = \sum_{q=0}^{L-1} q p_q(r_q)$$

函数 graythresh 取一幅图像，计算它的直方图，找到最大化σ_B^2的阈值。阈值返回为0.0和1.0之间的归一化值。函数 graythresh 的调用语法为

```
T = graythresh(f)
```

其中，f是输入图像，T是产生的阈值。为了分割图像，我们在函数im2bw中使用阈值T，函数im2bw已在2.7.2节中介绍。因为阈值已被归一化到范围[0, 1]内，因此必须在使用阈值之前将其缩放到合适的范围。例如，若 f 是 uint8 类图像，则我们在使用 T 之前要让 T 乘以 255。

例 10.7 计算全局阈值

在这个例子中，我们将说明先前描述过的迭代过程以及针对灰度图像 f 的Otsu方法，图像f是一幅扫描的文本图像，如图 10.13(a)所示。迭代方法可按如下方式实现：

```
>> T = 0.5*(double(min(f(:))) + double(max(f(:))));
>> done = false;
>> while ~done
     g = f >= T;
     Tnext = 0.5*(mean(f(g)) + mean(f(~g)));
     done = abs(T - Tnext) < 0.5;
     T = Tnext;
end
```

对这幅特殊图像来说，while 循环执行4次并在 T 等于 101.47 时结束。

接下来，我们使用函数 graythresh 来计算一个阈值：

```
>> T2 = graythresh(f)
```

```
T2 =
    0.3961
>> T2 * 255
ans =
    101
```

使用这两个值的阈值处理会产生彼此之间几乎没有差别的图像。图10.13(b)显示了使用 T2 进行阈值处理后的图像。



(a)　　(b)

图10.13　(a)扫描的文本；(b)使用函数 graythresh 得到的经阈值处理后的文本

10.3.2　局部阈值处理

如图9.26(a)和图9.26(b)所示的那样，全局阈值处理方法在背景照明不均匀时有可能无效。在这种情况下，一种常用的处理方法是针对照明问题做预处理以补偿图像，然后再对预处理后的图像采用全局阈值处理。图9.26(e)中显示的改进的阈值处理结果是通过应用一个形态学顶帽算子并对得到的结果使用函数 graythresh 来计算的。我们可以证明这种处理等同于使用局部变化的阈值函数 $T(x, y)$ 对 $f(x, y)$ 进行阈值处理：

$$g(x, y) = \begin{cases} 1 & 若 f(x, y) \geqslant T(x, y) \\ 0 & 若 f(x, y) < T(x, y) \end{cases}$$

其中，

$$T(x, y) = f_o(x, y) + T_o$$

图像 $f_o(x, y)$ 是 f 的形态学开运算，常数 T_o 是对 f_o 应用函数 graythresh 后的结果。

10.4　基于区域的分割

分割的目的是把图像分成区域。在10.1节和10.2节中，我们基于灰度级的不连续性来查找区域间的边界的方法，解决了这一问题。但在10.3节中，分割是通过基于像素属性的分布（如亮度值）的阈值来完成的。在这一节中，我们将讨论直接寻找区域的分割技术。

10.4.1　基础公式

用 R 表示整个图像区域。我们可将分割看做是把 R 分成 n 个子区域 $R_1, R_2, R_3, \cdots, R_n$ 的处理，满足：

(a) $\bigcup_{i=1}^{n} R_i = R$。

(b) R_i 是一个连接区域，$i = 1, 2, \cdots, n$。

(c) $R_i \cap R_j = \varnothing$，对所有的 i 和 j, $i \neq j$。

(d) $P(R_i) = \text{TRUE}$，对 $i = 1, 2, \cdots, n$。

(e) $P(R_i \cup R_j) = \text{FALSE}$，对任何邻接区域 R_i 和 R_j。

其中，$P(R_i)$ 是定义在集合 R_i 中的点上的逻辑谓词，$\varnothing$ 是空集。

条件(a)指出分割必须是完全的，即每个点都必须在一个区域中。条件(b)要求区域中的点应该是按预先定义好的方式（如4连接或8连接）连接的。条件(c)说明区域之间不相交。条件(d)说明分割区域中的像素点必须满足的性质——例如，若所有 R_i 中的像素有相同的灰度级，则 $P(R_i)$ = TRUE。最后，条件(e)指出，邻近区域 R_i 和 R_j 在谓词 P 上的意义是不一样的。

10.4.2 区域生长

顾名思义，区域生长是根据预先定义的生长准则来把像素或子区域集合成较大区域的处理方法。基本处理方法是以一组“种子”点开始来形成生长区域，即将那些预定义属性类似于种子的邻域像素附加到每个种子上（如指定的灰度级或颜色）。

由一个或多个开始点组成的集合的选择通常可基于问题的性质，如后面的例10.8所示。当没有先验的信息可用时，一种处理方法是在每一个像素上计算同一组属性，在生长过程期间，这些属性最终将用于把像素分配到区域中。若这些计算的结果显示了一簇值，则应把具有这些特性的像素放在可以作为种子的这些簇的质心。

相似性准则的选择不但依赖于所考虑的问题，而且也依赖于可用的图像数据类型。例如，利用人造卫星图像进行陆地分析会严重依赖于颜色的使用。在彩色图像中，若没有内在信息可用时，这个问题可能会更困难或根本无法解决。当图像为单色图像时，图像分析应该用一组基于灰度级（如纹理）和空间性质的描述符来执行。我们将在第11章中讨论对区域特征有用的描述符。

若在区域生长处理中未使用连通性（邻接）信息，则描述符会产生错误的结果。例如，仅使用三个不同的亮度值来显示像素的一个随机排列。若不考虑连通性而组合有着相同灰度级的像素以形成一个“区域”，则会产生无意义的分割结果。

区域生长中的另一个问题是停止规则的公式表达。一般来说，当不再有像素满足该区域所包含的准则时，生长区域的过程就会停止。准则（如亮度值、纹理、色彩）实际上是局部的，在区域生长的“历史”中不予考虑。增加区域生长算法能力的附加准则利用了大小的概念，如被生长像素和候选像素之间的相似性（如候选像素的亮度与生长区域的平均亮度的比较），正生长区域的形状等。这些类型的描述符的应用基于这样一个假设，即期望结果的模型至少是部分可用的。

为示例在MATLAB中是如何处理区域分割的，我们开发一个名为 `regiongrow` 的M函数来完成基本的区域生长。该函数的语法为

```
[g, NR, SI, TI] = regiongrow(f, S, T)
```

其中，`f` 是将被分割的图像，参数 `S` 可以是一个数组（与 `f` 大小相同）或一个标量。若 `S` 是一个数组，则它在所有种子点的坐标处必须为1，而在其他位置为0。这样的数组可以通过检测确定，或通过外部的种子查找函数来确定。若 `S` 是一个标量，则它定义一个亮度值，`f` 中有着该值的所有点都将变成种子。同样，`T` 也可以是数组（与 `f` 大小相同）或标量。若 `T` 是数组，则它会为 `f` 中的每个位置包含一个阈值。若 `T` 是标量，则它会定义一个全局阈值。阈值用来测试图像中的像素是否与该种子或8连接种子足够相似。

例如，若 `S = a` 且 `T = b`，则我们将比较亮度，若像素亮度和 `a` 之间的差的绝对值小于或等于 `b`，则我们称该像素类似于 `a`（在通过阈值测试的意义下）。另外，若问题中的像素是8连接到一个或多个种子值的，则这个像素就可看做是一个或多个区域的成员。若 `S` 或 `T` 是数组，则也有类似的结论，其基本差异是用 `S` 中定义的合适位置和相应的 `T` 值来比较。

在输出中，`g` 是分割后的图像，每个区域的成员都用整数标出。参数 `NR` 是不同区域的数目。参数 `SI` 是一幅包含有种子点的图像，参数 `TI` 是一幅图像，该图像中包含在经过连通性处理前通过阈值测试的像素。`SI` 和 `TI` 的大小均与 `f` 相同。

及255和左边第一个主谷的位置之间的差为基础的，主谷的位置表示黑色焊接区域中的最高亮度值。如早些时候说明的那样，在一个区域内至少有一个像素与将要被包含的另一个区域内的任何一个像素是8连接的。若发现一个像素已连接到多个区域上，则这些区域会自动地使用函数 regiongrow 来合并。

图 10.14(b)显示了种子点（图像 SI）。在这个例子中，种子点很多，因为种子被简单地指定为具有数值255的所有点。图 10.14(c)是图像 TI，它显示了所有通过阈值测试的点，即具有亮度 z_i 且满足 $|z_i - S| \leq T$ 的点。图 10.14(d)显示了提取图 10.14(c)中的所有连接到种子点的像素的结果。这是分割后的图像 g。很明显，通过比较这幅图像与原图像，区域生长确实以合理的精度分割了有缺陷的焊接。

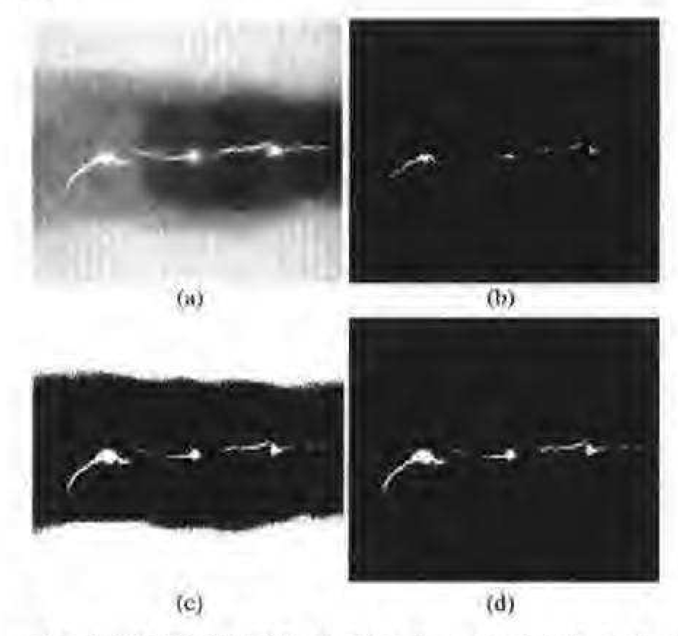


图 10.14　(a)显示有缺陷焊接的图像；(b)种子点；(c)显示所有已通过阈值测试的像素的二值图像（白色）；(d)图(c)中的所有像素在对种子点进行 8 连通性分析后的结果（原图像由 XTEK 系统有限公司提供）

最后，通过观察图 10.15 所示的直方图，我们注意到不可能通过任何 10.3 节中讨论的阈值方法来获得相同或等价的解决方案。在这种情况下，连通性是一个基本的要求。

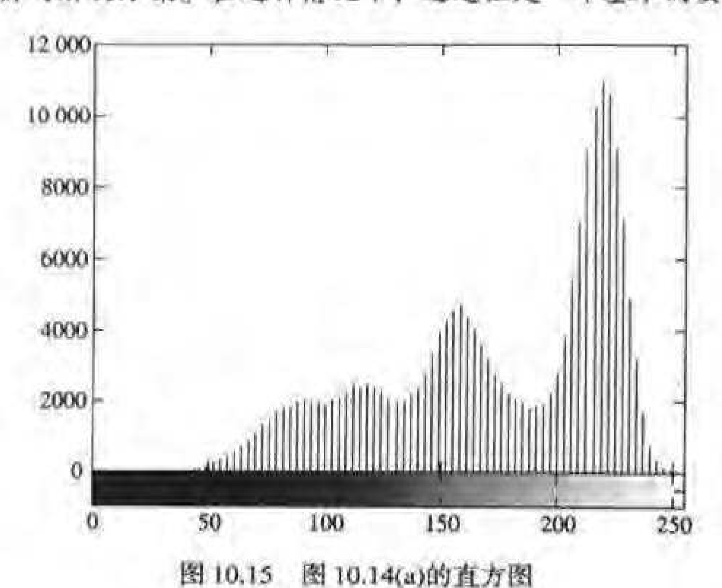


图 10.15　图 10.14(a)的直方图

10.4.3　区域分离和合并

我们刚刚讨论过从一组种子点来生长区域的过程。另一种方法是再把图像细分为一组任意且互不相连的区域，然后在试图满足10.4.1节规定的条件下合并或分离这些区域。分离和合并的基本原理将在稍后讨论。

令 R代表整个图像区域并选择一个谓词P。分割R的一种方法是把它再细分为更小的象限区域，以便对任何区域 R_i 都有 $P(R_i)$ = TRUE。我们从整个区域开始。若 $P(R)$ = FALSE，则把图像分成象限。若对任何一个象限来说 P 都是 FALSE，则再把象限细分为子象限，依次类推。这种特别的分离技术有一种方便的表示方法——四叉树，即一棵树，树中的每一个节点都恰好有四个后代，如图10.16所示（对应于四叉树的节点有时称为四叉区域或四叉图像）。注意，树的根对应于整个图像，每个节点对应于一个再分为四个后代节点的节点分支。在这种情况下，只有 R_4 被进一步细分了。

若仅使用了分离，则最终的部分通常包含具有相同性质的邻近区域。这种缺点可通过合并以及分离来弥补。若要满足10.4.1节中的约束条件，则要求只合并邻近区域，组合的像素满足谓词 P。换言之，仅当$P(R_j \cup R_k)$ = TRUE时，两个邻近区域R_j 和 R_k 才会合并。

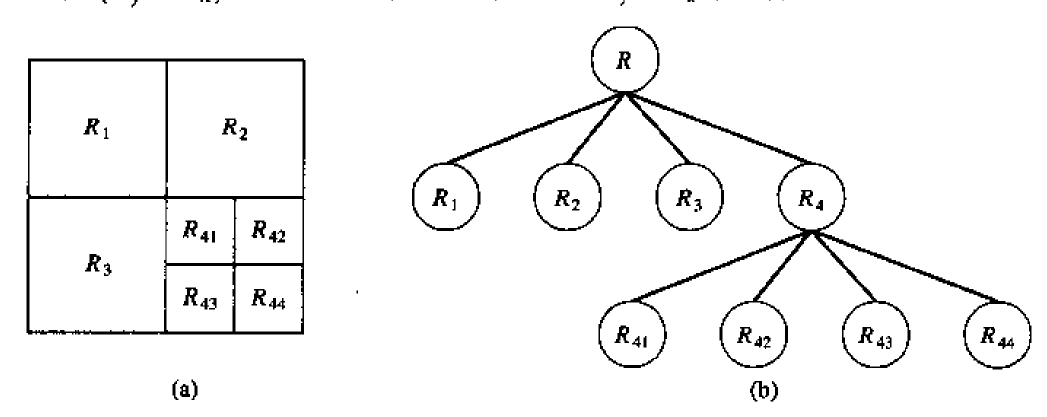


图10.16　(a)已分区的图像；(b)相应的四叉树

前面的讨论可总结为如下的步骤：

1. 把任意区域 R_i 分为4个不相连的象限，其满足 $P(R_i)$ = FALSE。
2. 当不可能再分时，合并任何满足$P(R_j \cup R_k)$ = TRUE 的区域R_j 和 R_k。
3. 发不可能进一步合并时，停止。

前面介绍的基本主题可能有多种变体。例如，若我们允许合并两个增满足谓词的邻近区域R_i和R_j，则会导致重要的简化。这会产生更为简单且更快的算法，因为谓词的测试受限于单个四叉区域。如例10.9所示，这种简化在实际中仍可产生较好的分割结果。在过程的步骤2中使用这一方法，所有满足谓词的四叉区域都将用1填充，并且它们的连通性可以很容易地检测出来，例如使用函数 `imreconstruct`。在效果上，该函数能完成邻近像限区域的期望合并。不满足谓词的四叉区域均用0填充，从而产生一幅分割的图像。

IPT中实现四叉树分解的函数是 `qtdecomp`。在本节中我们关注的语法形式为

```
S = qtdecomp(f, @split_test, parameters)
```

其中，`f`是输入图像，`S`是包含四叉树结构的稀疏矩阵。若`S(k,m)`非零，则`(k,m)`是分解中的一个左上角块，且该块的大小是`S(k,m)`。函数`split_test`（见下例中的函数`splitmerge`）用来

确定一个区域是否被分离，parameters（用逗号分开）是函数 split_test 所要求的附加参数。这种机理和 3.4.2 节中所讨论的函数 coltfilt 相似。

为了在四叉树分解中得到实际的四叉区域像素值，我们使用函数 qtgetblk，其语法为

```
[vals, r, c] = qtgetblk(f, s, m)
```

其中，vals 是一个数组，它包含 f 的四叉树分解中大小为 m × m 的块的值，S 是由 qtdecomp 返回的稀疏矩阵。参数 r 和 c 是包含左上角块的行坐标和列坐标的向量。

我们通过编写一个基本的分离和合并M函数来说明函数qtdecomp的用法，该函数使用前面讨论的简化，若其中的两个区域均满足谓词，则这两个区域将被合并。我们所调用的函数splitmerge的语法为

```
g = splitmerge(f, mindim, @predicate)
```

其中，f是输入图像，g是输出图像，其中的每一个连接区域都用不同的整数来标注。参数mindim定义分解中所允许的最小块；该参数必须是 2 的正整数次幂。

函数 predicate 是一个用户定义的函数，它必须包括在 MATLAB 路径中。其语法为

```
flag = predicate(region)
```

若 region 中的像素满足函数中由代码所定义的谓词，则函数就必须被写成返回 true（逻辑 1）；否则，flag 的值就必须为 false（逻辑 0）。例 10.9 示例了该函数的用法。

函数 splitmerge 的结构很简单。首先，图像被函数 qtdecomp 分块。函数 split_test 使用predicate确定区域是否应被分离。因为当区域被一分为四时，我们并不知道所产生的四个区域中的哪一个将通过谓词测试，在了解已分区图像中的哪个区域通过测试后，有必要检查区域。函数predicate也用于该目的。任何一个通过测试的四叉区域都用 1 填充，任何没有通过测试的四叉区域则用 0 填充。通过选择用 1 填充的每个区域中的一个元素，创建了一个标记数组。该数组与分区后的图像一起用于确定区域的连通性（邻接性）；函数 imreconstruct 用于该目的。

函数 splitmerge 的代码如下。代码注释部分中显示的简单谓词函数已用于例 10.9 中。注意，输入图像的大小已被增大为一个方形，其维数是可包围图像的 2 的最小整数次幂。这是函数 qtdecomp 所要求的，以确保分离为 1 是可能的。

```
function g = splitmerge(f, mindim, fun)
%SPLITMERGE Segment an image using a split-and-merge algorithm.
%   G = SPLITMERGE(F, MINDIM, @PREDICATE) segments image F by using a
%   split-and-merge approach based on quadtree decomposition. MINDIM
%   (a positive integer power of 2) specifies the minimum dimension
%   of the quadtree regions (subimages) allowed. If necessary, the
%   program pads the input image with zeros to the nearest square
%   size that is an integer power of 2. This guarantees that the
%   algorithm used in the quadtree decomposition will be able to
%   split the image down to blocks of size 1-by-1. The result is
%   cropped back to the original size of the input image. In the
%   output, G, each connected region is labeled with a different
%   integer.
%
%   Note that in the function call we use @PREDICATE for the value of
%   fun. PREDICATE is a function in the MATLAB path, provided by the
%   user. Its syntax is
%
```

```
%       FLAG = PREDICATE(REGION) which must return TRUE if the pixels
%       in REGION satisfy the predicate defined by the code in the
%       function; otherwise, the value of FLAG must be FALSE.
%
%    The following simple example of function PREDICATE is used in
     Example 10.9 of the book. It sets FLAG to TRUE if the
%    intensities of the pixels in REGION have a standard deviation
%    that exceeds 10, and their mean intensity is between 0 and 125.
%    Otherwise FLAG is set to false.
%
%       function flag = predicate(region)
%       sd = std2(region);
%       m = mean2(region);
%       flag = (sd > 10) & (m > 0) & (m < 125);

% Pad image with zeros to guarantee that function qtdecomp will
% split regions down to size 1-by-1.
Q = 2^nextpow2(max(size(f)));
[M, N] = size(f);
f = padarray(f, [Q - M, Q - N], 'post');

% Perform splitting first.
S = qtdecomp(f, @split_test, mindim, fun);

% Now merge by looking at each quadregion and setting all its
% elements to 1 if the block satisfies the predicate.

% Get the size of the largest block. Use full because S is sparse.
Lmax = full(max(S(:)));
% Set the output image initially to all zeros. The MARKER array is
% used later to establish connectivity.
g = zeros(size(f));
MARKER = zeros(size(f));
% Begin the merging stage.
for K = 1:Lmax
   [vals, r, c] = qtgetblk(f, S, K);
   if ~isempty(vals)
      % Check the predicate for each of the regions
      % of size K-by-K with coordinates given by vectors
      % r and c.
      for I = 1:length(r)
         xlow = r(I); ylow = c(I);
         xhigh = xlow + K - 1; yhigh = ylow + K - 1;
         region = f(xlow:xhigh, ylow:yhigh);
         flag = feval(fun, region);
         if flag
            g(xlow:xhigh, ylow:yhigh) = 1;
            MARKER(xlow, ylow) = 1;
         end
      end
   end
end
% Finally, obtain each connected region and label it with a
```

```
% different integer value using function bwlabel.
g = bwlabel(imreconstruct(MARKER, g));

% Crop and exit
g = g(1:M, 1:N);

%-------------------------------------------------------------------------%
function v = split_test(B, mindim, fun)
% THIS FUNCTION IS PART OF FUNCTION SPLIT-MERGE. IT DETERMINES
% WHETHER QUADREGIONS ARE SPLIT. The function returns in v
% logical 1s (TRUE) for the blocks that should be split and
% logical 0s (FALSE) for those that should not.

% Quadregion B, passed by qtdecomp, is the current decomposition of
% the image into k blocks of size m-by-m.

% k is the number of regions in B at this point in the procedure.
k = size(B, 3);

% Perform the split test on each block. If the predicate function
% (fun) returns TRUE, the region is split, so we set the appropriate
% element of v to TRUE. Else, the appropriate element of v is set to
% FALSE.
v(1:k) = false;
for I = 1:k
   quadregion = B(:, :, I);
   if size(quadregion, 1) <= mindim
      v(I) = false;
      continue
   end
   flag = feval(fun, quadregion);
   if flag
      v(I) = true;
   end
end
```

例 10.9 使用区域分离和合并的图像分割

图 10.17(a)显示了天鹅星座环的一幅X射线频段图像。图像的大小为 256 × 256 像素。该例的目的是分割出环绕致密中心的稀疏环。我们感兴趣的区域有某些明显的特征，这些特征在分割中会很有帮助。首先，我们注意到数据具有随机性，这表明稀疏环的标准偏差要比背景（值为0）和较大中心区域的标准偏差大。同样，包含外环区域数据的均值（平均亮度）应该比背景（值为0）的均值大，而比较大且较亮的中心区域的均值小。这样，我们应该能够使用这两个参数来分割感兴趣的区域。事实上，函数 splitmerge 的文档中的一个示例表明该谓词函数包含有该问题的一些知识。参数可通过计算图 10.17(a)中各个区域的均值和标准偏差来确定。图 10.17(b)至图 10.17(f)显示了使用函数 splitmerge 且 mindim 的值分别为 32, 16, 8, 4, 和 2 时进行分割后的结果。所有图像均显示了带有细节的分割结果，这些结果反比于mindim的值。图 10.17 中的所有结果都是合理的分割。若将这些图像之一用做掩模来提取原图像中的某个我们感兴趣的区域，则图 10.17(d)所示的结果可能是最好的选择，因为它是一个具有最多细节的实心区域。刚刚说明的方法的一个重要方面是它在函数 predicate 中“捕获”信息的能力，这种能力无疑问会有助于分割。

1	1	0	0	0
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0
0	1	1	1	0

(a)

0.00	0.00	1.00	2.00	3.00
0.00	0.00	1.00	2.00	3.00
1.00	1.00	1.41	2.00	2.24
1.41	1.00	1.00	1.00	1.41
1.00	0.00	0.00	0.00	1.00

(b)

图 10.19 (a)小二值图像；(b)距离变换

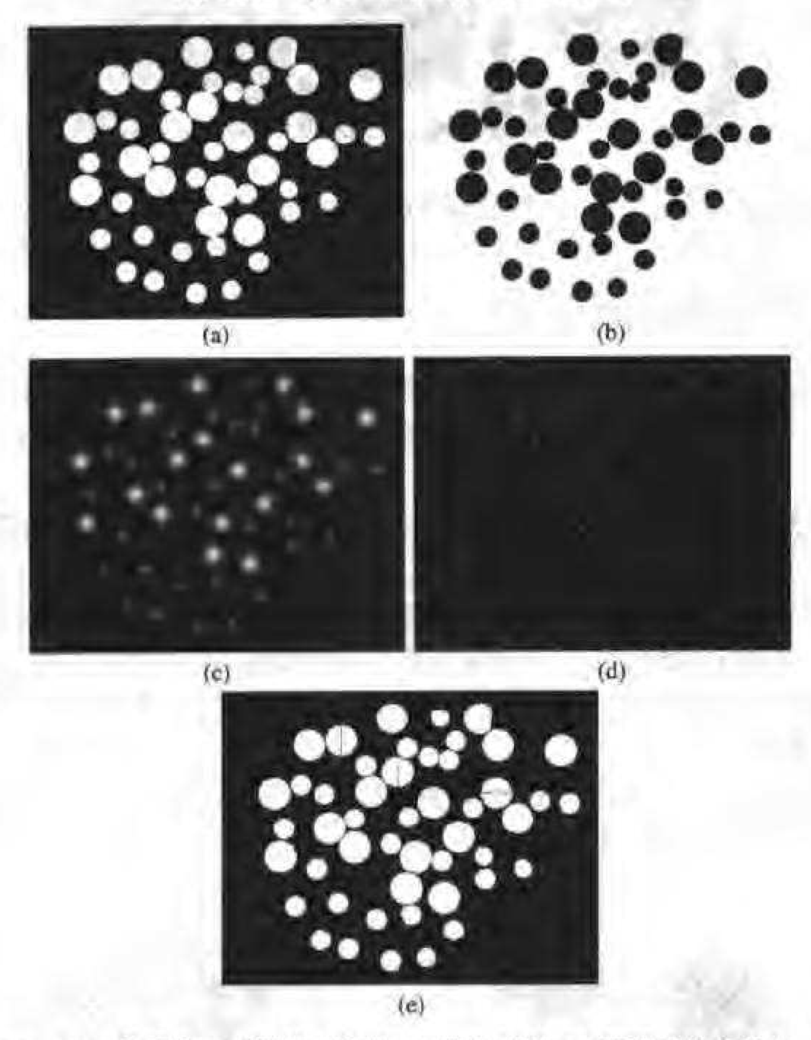


图 10.20 (a)二值图像；(b)图像(a)的补；(c)距离变换；(d)距离变换的负分水岭脊线；(e)黑色叠加在原始二值图像上后的分水岭脊线

例 10.10 用距离和分水岭变换分割二值图像

在这个例子中，我们将说明如何结合使用IPT的分水岭变换与距离变换来分割圆形水滴。特别地，我们想要分割如图 9.29(b)所示的已处理按钉图像 `f`。首先，如 10.3.1 节描述的那样，我们使用函数 `im2bw` 和 `graythresh` 将图像变换为二值图像。

```
>> g = im2bw(f, graythresh(f));
```

图 10.20(a)显示了结果。下一步是对图像求补，计算其距离变换，然后使用函数 `watershed` 来计算距离变换的负分水岭变换。该函数的调用语法为

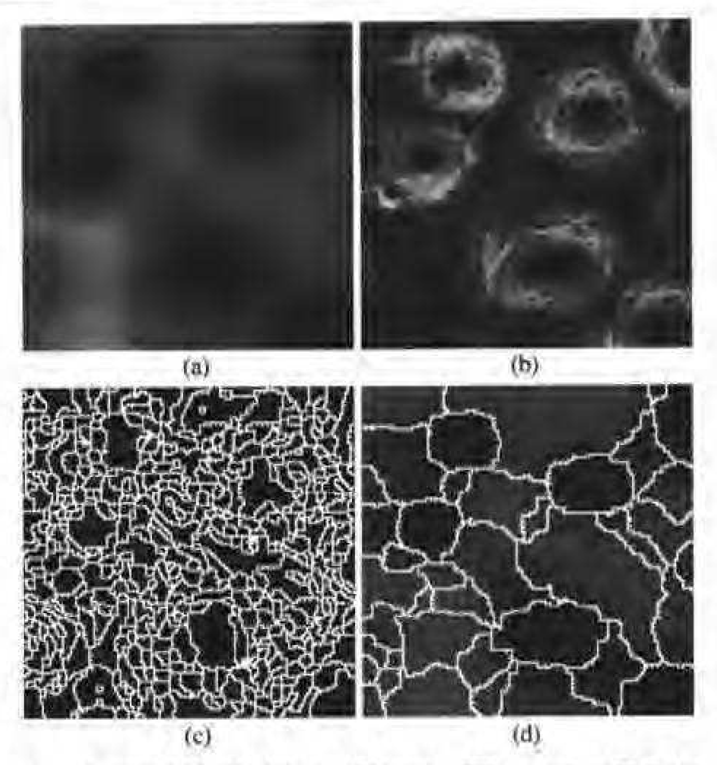


图 10.21 (a)小气泡的灰度级图像；(b)梯度幅度图像；(c)显示了几个过分割的图像(b)的分水岭变换；(d)平滑梯度图像后的分水岭变换，一些过分割仍然比较明显（原图像由巴黎 CMM/Ecole 的 S. Beucher 博士提供）

10.5.3 控制标记符的分水岭分割

分水岭变换直接用于梯度图像时，噪声和梯度的其他局部不规则性常常会导致过分割。其导致的问题可能会非常严重，以至于产生不可用的结果。按照现在的思路，这将意味着具有大量的分割区域。解决该问题的一种实际方法是把加入一个预处理阶段，以将其他知识带到分割过程中，从而限制允许的区域数目。

用于控制过分割的一种方法基于标记符的概念。标记符是一个属于一幅图像的连接分量。我们希望有一个内部标记符集合（处在每一个感兴趣对象的内部）和一个外部标记符集合（包含在背景中）。这些标记符然后使用例10.12中描述的过程来修改梯度图像。用于计算内部和外部标记符的方法有许多，其中包括前面描述的线性滤波、非线性滤波及形态学处理。对于特定的应用，我们选择的方法高度依赖于与应用相关的图像的特性。

例 10.12 标记符控制的分水岭分割示例

在这个例子中，我们将把标记符控制的分水岭分割用于图 10.22(a)中的电泳凝胶图像。我们从考虑计算梯度图像的分水岭变换得到的结果开始，并且不做任何其他处理。

```
>> h = fspecial('sobel');
>> fd = double(f);
>> g = sqrt(imfilter(fd, h, 'replicate') .^ 2 + ...
            imfilter(fd, h', 'replicate') .^ 2);
>> L = watershed(g);
>> wr = L == 0;
```

从图 10.22(b)中我们可以看到部分由大量局部最小区域导致的过分割结果。IPT 函数 imregionalmin 计算图像中大量局部最小区域的位置，其调用语法为

```
rm = imregionalmin(f)
```

其中，f是灰度图像，rm是二值图像，二值图像的前景像素标记了局部最小区域的位置。我们可对梯度图像使用函数imregionalmin，从而了解为什么函数watershed会产生如此多的小汇水盆地：

```
>> rm = imregionalmin(g);
```

图10.22(c)中显示的多数局部最小区域位置非常浅，并且表示了与我们的分割问题不相关的细节。为消除这些无关的小区域，我们使用IPT函数imextendedmin，该函数可计算图像中的"低点"集合，即比周围点更深的点的集合（通过某个高度阈值）。扩展的最小变换和相关操作的详细内容，请参阅Soille[2003]。该函数的调用语法为

```
im = imextendedmin(f, h)
```

其中，f是一幅灰度级图像，h是高度阈值，im是一幅二值图像，该二值图像的前景像素标记了深局部最小区域的位置。这里，我们使用函数imextendedmin来获得内部标记符集合：

```
>> im = imextendedmin(f, 2);
>> fim = f;
>> fim(im) = 175;
```

最后两行将在原图像上以灰色气泡的形式叠加扩展的局部最小区域位置，如图10.22(d)所示。我们看到，得到的气泡确实很合理地标记了我们想要分割的对象。

接下来，我们必须寻找外部标记符或者我们确信属于背景的像素。我们在这里采用的方法是用找到的像素去标记背景，这些像素恰好位于内部标记符间的中间位置。令人惊讶的是，通过求解另一个分水岭问题，我们可做到这一点；特别地，我们可计算内部标记符图像im的距离变换的分水岭变换：

```
>> Lim = watershed(bwdist(im));
>> em = Lim == 0;
```

图10.22(e)显显示了二值图像em中的分水岭脊线。因为这些脊线位于由im标记的暗气泡间的中间位置，所以它们应该是很好的外部标记符。

给出内部和外部标记符后，然后就可使用它们来修改梯度图像，方法是使用称为强制最小（minima imposition）的过程。强制最小技术（详见Soille[2003]）修改了灰度级图像，以便局部最小区域仅出现在标记的位置。其他像素值将按需要"上推"，以便删除其他的局部最小区域。IPT函数imimposemin可实现这种技术。该函数的调用语法为

```
mp = imimposemin(f, mask)
```

其中，f是一幅灰度级图像，mask是一幅二值图像，该二值图像的前景像素标记出了输出图像mp中局部最小区域的期望位置。通过在内部和外部标记符的位置覆盖局部最小区域，我们可修改梯度图像：

```
>> g2 = imimposemin(g, im | em);
```

图10.22(f)显示了结果。最后，我们准备计算修改了标记符的梯度图像的分水岭变换，并研究所导致的分水岭脊线：

```
>> L2 = watershed(g2);
```

```
>> f2 = f;
>> f2(L2 == 0) = 255;
```

最后两行会在原图像上叠加分水岭脊线。有着很大改进的分割结果如图10.22(g)所示。

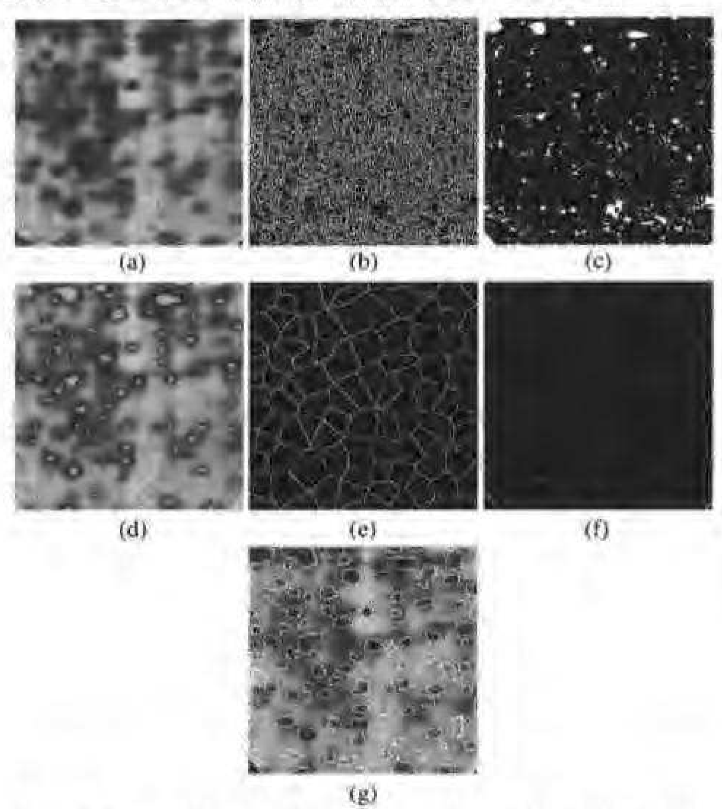


图10.22 (a)凝胶图像；(b)对梯度幅值图像应用分水岭变换后的过分割结果；(c)梯度幅值的局部最小区域；(d)内部标记符；(e)外部标记符；(f)修改后的梯度幅值；(g)分割结果（原图像由巴黎CMM/Ecole的S. Beucher博士提供）

标记符的选择范围可以从刚才描述的简单过程到涉及尺寸、形状、位置、相对距离、纹理内容等的复杂方法（见第11章中关于描绘子的内容）。要点是使用标记符会为分割问题带来先验知识。人们常常使用先验知识来帮助解决分割和更高级的任务。因此，分水岭分割可提供一个充分利用这种知识的框架的事实是这一方法的突出优点。

小结

在大多数自动图像模式识别和场景分析问题中，图像分割是一个基本的预备步骤。正如本章中的例子所示的那样，选择哪一种分割技术将由所考虑问题的特殊特性决定。本章所讨论的方法虽然不够彻底，但基本代表了实际中常用的技术。

第11章 表示与描述

前言

在使用诸如第10章讨论的方法将一幅图像分割为区域后，接下来通常要对分割区域加以表示与描述，以便使“自然状态的”像素更适合计算机处理。表示区域涉及到两个基本选择：(1)用外部特征（区域的边界）表示区域；(2)用内部特征（组成区域的像素）表示区域。然而，选择一种表示方案仅仅是使数据更适宜于计算机处理的任务的一部分。下一个任务是在选择了表示方案的基础上描述区域。例如，区域可以用边界来表示，而边界可以用诸如边界长度和其包含的凹面形状的数目等特征来描述。

当我们对形状特征感兴趣时，可以选择外部表示；而当我们的主要注意力集中于区域属性时，则可以选择内部表示，如颜色和纹理。有时可用这两种表示来解决同一问题。无论哪一种情况，被选做描绘子的特征应该尽可能对区域大小、平移与旋转的变化不敏感。在很大程度上，本章讨论的描绘子满足一个或者多个这样的属性。

11.1 背景知识

区域是一个连接的分量，而区域的边界（也称为边框或轮廓）则是区域像素的集合，这些像素有一个或多个不在区域内的相邻像素。不在边界或区域上的点用0来表示，称为背景点。最初，我们感兴趣的仅仅是二值图像，因此，区域或者边界点用1来表示，而背景点则用0来表示。在本章后面，我们允许像素具有灰度级值或多谱值。

由前一段给出的定义，我们可得出以下结论：边界是一组相连的点。若边界上的点形成一个顺时针或者逆时针序列，则称边界上的点为有序的。若边界上的每个点均恰好有两个值为1的相邻像素点，而且这些邻像素点不是4邻接的，则称边界是最低限度连接的。内部点定义为区域内除边界外的任意位置上的点。

在某种意义上，本章内容与迄今为止所讨论的内容大不相同，因此，我们必须能够处理不同类型的数据，如边界、区域、拓扑数据等。接下来，我们将简要介绍一下在本章后面会用到的一些基本MATLAB和IPT概念与函数。

11.1.1 单元数组与结构

我们先从MATLAB的单元数组和结构开始讨论，它们在2.10.6节中已简要介绍过。

单元数组

单元数组提供了一种将各种类型的对象（如数字、字符、矩阵和其他单元数组）组合在一个变量名下的方法。例如，假设我们要处理下列三个实体：(1)一幅大小为512 × 512的`uint8`类图像`f`；(2)一个以188 × 2数组的行的形式出现的二维坐标序列`b`；(3)一个包含两个字符名称`char_array = {'area','centroid'}`的单元数组。这三个相异实体可以使用单元数组组织到一个变量`C`中：

```
C = {f, b, char_array}
```

例 11.1　单元数组的简单说明

假设我们要写一个函数，该函数输出一幅图像的平均亮度、维数、行的平均亮度和列的平均亮度，则可以使用“标准”方法按以下形式写函数：

```
function [AI, dim, AIrows, AIcols] = image_stats(f)
dim = size(f);
AI = mean2(f);
AIrows = mean(f, 2);
AIcols = mean(f, 1);
```

其中 f 为输入图像，输出变量对应于上面提及的量。若使用单元数组，则可写为

```
function G = image_stats(f)
G{1} = size(f);
G{2} = mean2(f);
G{3} = mean(f, 2);
G{4} = mean(f, 1);
```

与其他变量语法类似，写为G(1) ={size(f)}也是可以接受的。单元数组可以是多维的。例如，前一个函数也可以写为

```
function H = image_stats2(f)
H(1, 1) = {size(f)};
H(1, 2) = {mean2(f)};
H(2, 1) = {mean(f, 2)};
H(2, 2) = {mean(f, 1)};
```

或者，对其他变量也可以使用形式H{1,1} =size(f)等。额外的维数可以按类似的形式操作。假设 f 的大小为 512 × 512。在提示符后键入 G 和 H，则有

```
>> G = image_stats(f)
>> H = image_stats2(f);
>> G

G =
    [1x2 double]    [1]    [512x1 double]    [1x512 double]
>> H

H =
    [   1x2 double]    [             1]
    [512x1 double]    [1x512 double]
```

若想对G中的任意变量进行操作，则可以像前面一样，通过对单元数组中的特定元素进行访问来提取它。例如，若想处理 f 的大小，则可以写为

```
>> v = G{1}
```

或

```
>> V = H{1, 1}
```

其中 v 是一个大小为 1 × 2 的向量。注意，这里并没有使用我们熟悉的命令[M,N] = G{1}来获得图像的大小。若使用该命令，则会产生错误，因为只有函数可以产生多个输出。为获得M和N，应使用 M = v(1) 和 N = v(2)。

当输出数很多时，前面例子中符号的节省将更加明显。相对于给输出分配名称，其中一个缺点就是使用数字访问会丧失清晰度。这一点可以使用结构来加以弥补。

结构

就允许将不同的数据收集在一起组成一个单一变量而言，结构类似于单元数组。然而，与单元数组不同的是，单元是通过数字来访问的，而结构的元素则是通过称为域的名称进行访问的。

例 11.2 结构的简单说明

继续使用例 11.1 的主题，可以阐明这些概念。使用结构时，我们可以写为

```
function s = image_stats(f)
s.dim = size(f);
s.AI = mean2(f);
s.AIrows = mean(f, 2);
s.AIcols = mean(f, 1);
```

其中 s 是一个结构。本例中结构的域是 AI（一个标量）、dim（一个 1×2 向量）、AIrows（一个 $M \times 1$ 向量）和 AIcols（一个 $1 \times N$ 向量），其中 M 和 N 分别是图像的行与列的大小。注意，点的作用是将结构与各种域分开。域名可以是任意的，但必须以非数字字符开始。

使用例 11.1 中的图像，在提示符后键入 s 和 size(s)，可得到如下输出：

```
>> s =
s =
       dim:  [512 512]
        AI:  1
    AIrows:  [512x1 double]
    AIcols:  [1x512 double]
>> size(s)
ans =
    1    1
```

注意，s 本身是一个标量，它与本例中的 4 个域相关。

我们可以看到，本例中代码的逻辑与前面相同，但输出数据的组织更加清楚。正如在单元数组的情形下那样，当我们处理大量的输出时，使用结构的优点将变得更加明显。

前面的例子使用的是一个单一结构。若不是一幅图像，而是由 Q 幅图像组成的一个大小为 $M \times N \times Q$ 的数组，则函数变为

```
function s = image_stats(f)
K = size(f);
for k = 1:K(3)
    s(k).dim = size(f(:, :, k));
    s(k).AI = mean2(f(:, :, k));
    s(k).AIrows = mean(f(:, :, k), 2);
    s(k).AIcols = mean(f(:, :, k), 1);
end
```

换言之，结构本身是可以被索引的。如单元数组那样，虽然结构可以有很多维数，但其最常见的形式就是一个向量，如上面的函数所示。

从域中提取数据需要知道结构 s 和域的维数。例如，接下来的代码获得域 AIrows 的所有值并保存到 v 中：

```
for k = 1:length(s)
    v(:, k) = s(k).AIrows;
end
```

注意，冒号(:)是v的第一维，而k是第二维，因为s的维数是$1 \times Q$，而AIrows的维数是$M \times Q$。因此，由于k是从1到Q，所以v的维数是$M \times Q$。若我们感兴趣的是提取AIcols的值，则可在循环中使用v(k,:)。

若一个结构的域包含标量，则使用方括号可以将提取的信息放入一个向量或者矩阵。例如，假设D.Area包含图像中20个区域的面积，则

```
>> w = [D.Area];
```

将创建一个大小为1×20的向量w，其每个元素是一个区域的面积。

就像单元数组那样，当一个值赋给一个结构域时，MATLAB将在结构中产生该值的一个副本。若后来原始值发生了变化，则这种变化不会反映到结构中。

11.1.2　本章中使用的其他一些 MATLAB 和 IPT 函数

函数imfill在表9.3中和9.5.2节中已简要提及过。该函数对于二值图像和灰度图像的作用不同。因此，为有助于在本节中澄清符号的含义，我们分别用fB和fI来表示二值图像和灰度图像。若输出是二值图像，则我们用gB来表示它；否则将简单地表示为g。语法

```
gB = imfill(fB, locations , conn)
```

在输入二值图像fB 的背景像素上从参数locations指定的点开始，执行填充操作(即将背景像素值设为1)。参数locations可以是一个$n \times 1$向量(n是位置的数目)，在这种情况下，向量包含起始坐标位置的线性索引(见2.8.2节)。参数locations也可以是一个$n \times 2$矩阵，矩阵中的每行包含有fB中的一个起始位置的二维坐标。参数conn指定背景像素使用的连接方式：4(默认)或8。若参数location和参数conn在输入参数中均省略了，则命令gB = imfill(fB)将在屏幕上显示二值图像fB，并让用户用鼠标选择起始位置。单击鼠标左键添加点。按下**BackSpace**键或者**Delete**键可删除前面选择的点。使用shift键加鼠标单击、鼠标右击或双击可选择最后一个点，然后就可以开始填充操作。按**Return**键可完成没有附加点的选择。

使用语法

```
gB = imfill(fB, conn, 'holes')
```

可填充输入二值图像中的孔洞。孔洞是一个背景像素集合，它不能通过图像边缘来填充背景达到这一目的。像以前一样，conn指定连接方式：4(默认)或8。

语法

```
g = imfill(fI, conn, 'holes')
```

将填充输入灰度图像fI的孔洞。在这种情况中，孔洞是指较亮像素包围的暗像素区域，参数conn如上所述。

函数find可以和bwlabel一起使用，返回构成某个指定对象的像素的坐标向量。例如，若[gB,num] = bwlabel(fB)产生了多个连接区域(即num> 1)，则使用以下语法可以获得第二个区域的坐标：

```
[r, c] = find(g == 2)
```

在本章中，区域或者边界的二维坐标被组织成$np \times 2$数组的形式，其中每行是一个(x, y)坐标对，np是区域或边界中的点的数目。在某些情况下，有必要对数组进行排序。为此，可使用函数sortrows：

向量 v 包含输入图像中最长边界的坐标，k 是对应的区域数；数组 v 的大小为 $np \times 2$。若最大边界多于一条，则最后一条命令简单地选出最大长度边界中的第一个。如前一段所述，因为使用函数 boundaries 计算出的每个边界中第一个点和最后一个点相同，所以行 v(1,:) 和行 v(end,:) 相同。

语法为

```
b8 = bound2eight(b)
```

的函数 bound2eight 会从 b 中去除一些像素（这些像素对 4 连接来说是必需的，但对 8 连接来说则不是必需的）而留下一个其像素仅是 8 连接的边界。输入 b 必须是一个大小为 $np \times 2$ 的矩阵，它的每一行包含一个边界像素的 (x, y) 坐标。函数 bound2eight 要求 b 是闭合的，连接的像素集合按顺时针方向或逆时针方向排列。相同的条件也适用于函数 bound2four：

```
b4 = bound2four(b)
```

该函数会在存在对角连接的位置插入新边界像素，从而产生其像素仅是 4 连接的输出边界。两个函数的代码清单可在附录 C 中找到。

函数

```
g = bound2im(b, M , N, x0, y0)
```

生成一幅二值图像 g，该图像的大小为 $M \times N$，边界点为 1，背景值为 0。参数 x0 和 y0 决定图像中 b 的最小 x 和 y 坐标位置。边界 b 必须是坐标的一个大小为 $np \times 2$（或 $2 \times np$）的数组，其中，像前面提到的那样，np 是点的数目。若 x0 和 y0 被省略，则在 $M \times N$ 数组中边界会被近似中心化。若 M 和 N 被省略，则图像的水平和垂直尺度就等于边界 b 的长度和宽度。若函数 boundaries 发现多个边界，则可以使用函数 bound2im，通过连接单元数组 B 的元素，获得对应的所有坐标：

```
b = cat(1, B{:})
```

其中参数 1 表示沿数组的第一（垂直）维进行级联。

函数

```
[s, su] = bsubsamp(b, gridsep)
```

在一个网格上对单一边界 b 二次取样，网格线由 gridsep 像素分离。输出 s 是一个比 b 有更少的点的边界，点的数目由 gridsep 的值确定，su 是按比例取得的边界点的集合，这样可使坐标的转换趋于一致。对于 11.1.2 节中讨论的使用链码对边界进行编码，这非常有用。要求 b 中的点按顺时针方向或逆时针方向排列。

当使用函数 bsubsamp 对边界进行二次取样时，它的点将不再连接在一起。我们可以使用如下命令将其重新连接起来：

```
z = connectpoly(s(:, 1), s(:, 2))
```

其中 s 的行是二次取样后的边界的坐标。要求 s 中的点不是按顺时针方向排列就是按逆时针方向排列。输出 z 的行是连接边界的坐标，它是通过连接 s 中的使用 4 连接或 8 连接直线段的最短可能路径的点而成的。该函数可用于产生一个全连接的多边形边界，该边界通常要比从 s 得到的原始边界 b 更平滑（和简单）。在使用函数（如 minperpoly）来生成一个多边形的顶点时，函数 connectpoly 也是相当有用的，详见 11.2.3 节中的讨论。

当对边界进行操作时（例如，函数 connectpoly 需要一个做这项操作的子函数），计算连接网点的一条直线的整数坐标是一个基本工具。IPT 函数 intline 可很好地适用于这一目的。其语法为

```
[x, y] = intline(x1, x2 , y1 y2)
```

其中(x1,y1)和(x2,y2)分别是两个待连接点的整数坐标。输出x和y是列向量，它包含连接两点的一条直线的*x*坐标和*y*坐标。

11.2 表示

正如本章开始提到的，在第10章中讨论的分割技术以像素的形式沿边界或包含在区域中的像素产生原始数据。虽然有时这些数据直接用于获得描绘子（如决定区域的纹理），但是习惯上使用的是将数据压缩为一种表示的方案，这些表示被认为在描绘子计算中更有用。本节我们将讨论各种表示方法的实现。

11.2.1 链码

链码通过一个指定长度与方向的直线段的连接序列来表示一条边界。典型情况下，这一表示建立在线段的4连接或者8连接之上。每条线段的方向通过一个编号方案加以编码，如图11.1(a)和图11.1(b)所示。基于这种方式的链码称为Freeman链码。

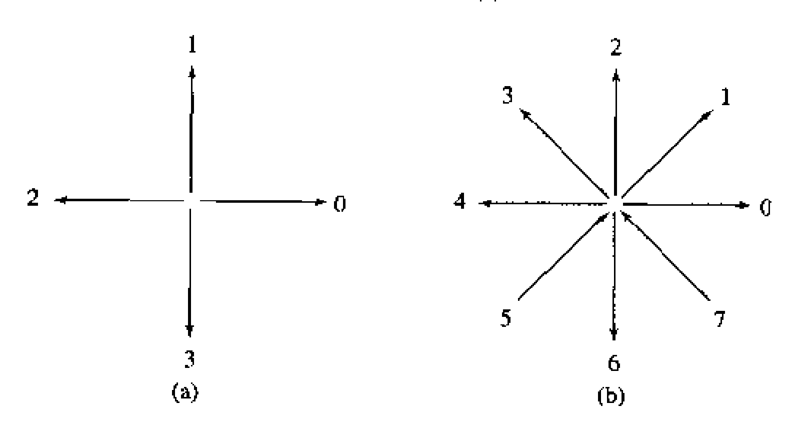


图11.1 (a)4方向链码的方向数；(b)8方向链码的方向数

一条边界的链码取决于起点。然而，代码可以通过将起点处理为方向数的循环序列和重新定义起点的方法进行归一化，因此，产生的数字序列形成一个最小幅值的整数。可以通过使用链码的一阶差分来代替链码本身的方法对旋转进行归一化[增量为90°或45°，如图11.1(a)和图11.1(b)所示]。这种差分可通过计算分隔链码的两个相邻像素的方向变化（图11.1中的逆时针方向）次数来获得。例如，4方向链码10103322的一阶差分为3133030。若我们将链码当做一个循环序列对待，则差分的第一元素可以通过使用链码的第一个和最后一个元素的转换加以计算。这里，结果是33133030。关于任意旋转角度的归一化，可以通过确定带有某些主要特性（如将在11.3.2节中讨论的主轴）的边界获得。

函数fchcode的语法为

```
c = fchcode(b, conn, dir)
```

该函数计算一个保存在数组b中的$np \times 2$个已排序边界点集的Freeman链码。输出c是一个包含以下域的结构，其中包含在圆括号中的数字表示数组的大小：

c.fcc = Freeman链码（$1 \times np$）
c.diff = 代码c.fcc的一阶差分（$1 \times np$）
c.mm = 最小幅度的整数（$1 \times np$）
c.diffmm = 代码c.mm的一阶差分（$1 \times np$）
c.x0y0 = 代码开始处的坐标（1×2）

参数conn指定代码的连接方式；其值可以是4或8（默认）。仅当边界不包含对角转换时，值设为4才是有效的。

参数dir指定输出代码的方向：若设定为'same'，则代码的方向与b中的点的方向相同。若设定为'reverse'，则代码的方向与b中的点的方向相反。默认为'same'。因此，写为c = fchcode(b,conn)时将使用默认方向，而写为c = fchcode(b)时则使用默认的连接方式和方向。

例 11.3 Freeman 链码及其某些变体

图11.2(a)显示一幅在反射噪声中嵌入的圆形斧子图像f。本例的目的是获得对象的边界的链码和一阶差分。从图11.2(a)中不难看出，对象上的噪声将导致一条很不规则的边界，它不是对象的一般形状的真实描述。当对噪声边界进行处理时，平滑是最常见的处理。图 11.2(b)显示了使用9 × 9平均掩模的处理结果g：

```
>> h = fspecial('average', 9);
>> g = imfilter(f, h, 'replicate');
```

图 11.2(c)所示的二值图像是经阈值处理后获得的：

```
>> g = im2bw(g, 0.5);
```

该图像的边界可使用上节讨论的函数boundaries来计算：

```
>> B = boundaries(g);
```

正如11.1.3节中说明的那样，我们感兴趣的是最长边界：

```
>> d = cellfun('length', B);
>> [max_d, k] = max(d);
>> b = B{1};
```

图 11.2(d)所示的边界图像是使用如下命令产生的：

```
>> [M N] = size(g);
>> g = bound2im(b, M, N, min(b(:, 1)), min(b(:, 2)));
```

直接获得b的链码将产生较小变化的长序列，边对于一般的图像形状表示是不必要的。因此，作为一种典型的链码处理，我们可以使用上节讨论的函数bsubsamp对边界进行二次取样：

```
>> [s, su] = bsubsamp(b, 50);
```

这里，我们使用了一个大约等于图像宽度10%的网格分离，此对图像的大小约为570 × 570像素。结果点可以显示为一幅图像［见图 11.2(e)］：

```
>> g2 = bound2im(s, M, N, min(s(:, 1)), min(s(:, 2)));
```

或使用如下命令显示为一个连接序列［见图 11.2(f)］：

```
>> cn = connectpoly(s(:, 1), s(:, 2));
>> g2 = bound2im(cn, M, N, min(cn(:, 1)), min(cn(:, 2)));
```

与图11.2(d)相反，为链编码使用边种表示的优点与图11.2(f)相比是显而易见的。链码可通过经过比例处理的序列su获得：

```
>> c = fchcode(su);
```

该命令将产生以下输出：

```
>> c.x0y0
ans =
    7    3
>> c.fcc
ans =
2 2 0 2 2 0 2 0 0 0 0 6 0 6 6 6 6 6 6 6 6 4 4 4 4 4 4 2 4 2 2 2
>> c.mm
ans =
0 0 0 0 6 0 6 6 6 6 6 6 6 6 4 4 4 4 4 4 2 4 2 2 2 2 2 0 2 2 0 2
>> c.diff
ans =
0 6 2 0 6 2 6 0 0 0 6 2 6 0 0 0 0 0 0 0 6 0 0 0 0 0 6 2 6 0 0 0
>> c.diffmm
ans =
0 0 0 6 2 6 0 0 0 0 0 0 0 6 0 0 0 0 0 0 6 2 6 0 0 0 0 0 6 2 0 6 2 6
```

通过检查c.fcc、图11.2(f)和c.x0y0，我们可以看出代码开始于图形的左边，并按顺时针方向处理，而该方向与边界的坐标相同。

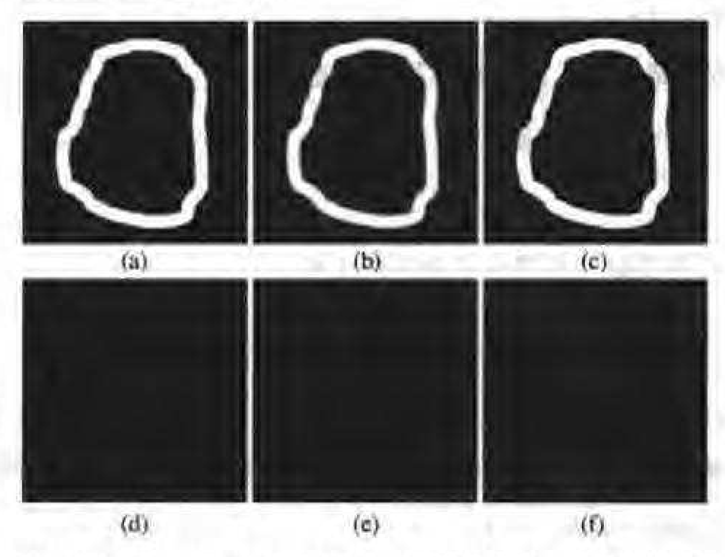


图11.2 (a)噪声图像；(b)用9×9平均掩模平滑后的图像；(c)经阈值处理后的图像；(d)二值图像的边界；(e)二次取样后的边界；(f)连接图(e)中的点后的效果

11.2.2 使用最小周长多边形的多边形近似

一条数字边界可由一个多边形以任意精度近似。对于一条闭合曲线，当多边形线段数目与边界点数目相同时，近似是精确的。因此，每一对相邻点定义了多边形的一条边。实践操作中，多边形近似的目的是用尽可能少的顶点表示边界的形状。

一种用于多边形近似的特殊方法是寻找一个区域或者一条边界的最小周长多边形（MPP）。其理论基础和寻找MPP的相应算法已在Sklansky等[1972]的经典论文中讨论过（也可以参阅Kim and Sklansky[1982]）。本节将给出算法的基本原理以及相应过程的M函数实现。该方法局限于“简单的”多边形（如本身不相交的多边形）。而且，只有一个像素宽度的突出区域将被排除在外。这些突出可以通过形态学方法加以提取，然后在计算多边形近似后再添加上去。

基础知识

首先，我们通过一个简单的例子来澄清一些概念。假设我们用一组级联的单元来包围一条边界，如图11.3(a)所示。这将有助于边界的可视化，就像对应于单元条的内外边界的两堵墙，对象边界可以被看成是包围两堵墙的区域间的橡皮条。若这一橡皮条可以收缩，则可得到如图11.3(b)所示的形状，由此产生一个与通过由元素条建立的几何体相符的最小周长多边形。

Sklansky 方法使用一个所谓的"细胞联合体"或者"细胞马赛克"，对我们来说，它是一组用于包围边界的方形元素的集合，如图11.3(a)所示。图11.4(a)显示了一个为细胞联合体所包围的区域（阴影区），注意，该区域的边界构成了一条4连接路径。若按顺时针行进，在凸角处（其内角为90°）我们用一个黑点(·)表示，在凹角处（其内角为270°）我们用一个白点（◦）表示。如图11.4(b)所示，黑点位于凸角本身上，白点则位于相应凹角的对角位置。这对应于算法的细胞联合体和顶点定义。

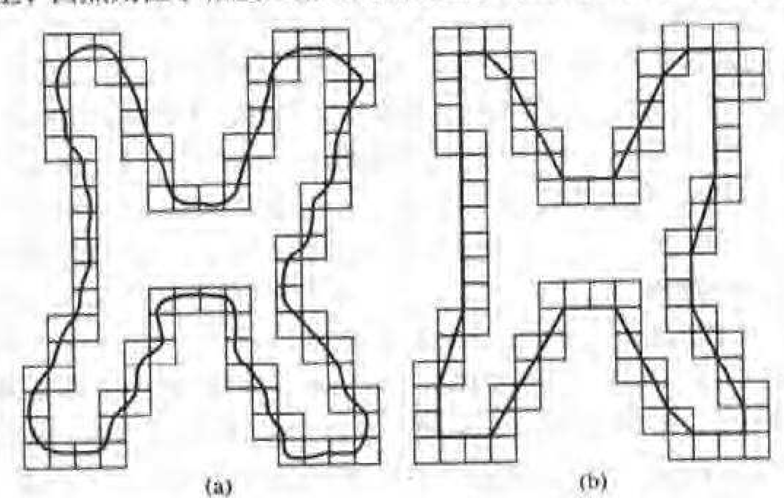


图 11.3　(a)由单元包围的对象边界；(b)最小周长多边形

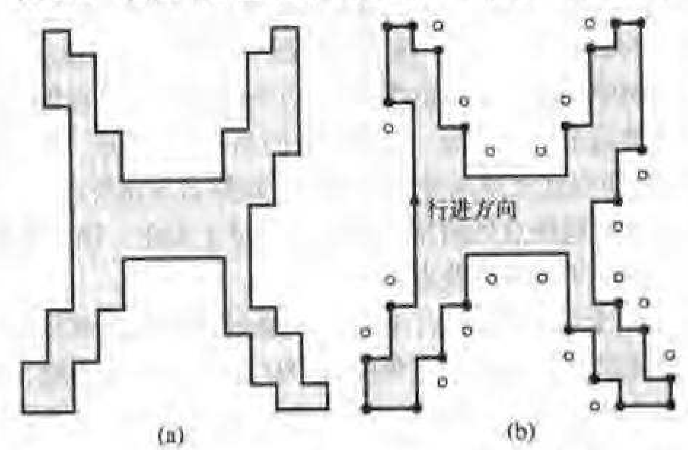


图 11.4　(a)图11.3(a)中由细胞联合体内墙包围的区域；(b)图(a)中区域的边界的凸角(·)和凹角(◦)标识符。注意，凹角标识符位于相应凹角的对角位置

以下特性是阐明寻找 MPP 的方法的基础：

1. 对应一个简单连接的细胞联合体的 MPP 是本身不相交的。令 P 表示该 MPP。
2. P 的每一个凸顶点与一个黑点 · 相符（但并不是所有黑点均是 P 的顶点）。
3. P 的每一个凹顶点与一个白点◦相符（但并不是所有白点均是 P 的顶点）。
4. 若一个黑点 · 为 P 的一部分，且不是 P 的凸顶点，则其在 P 的边缘上。

在我们的讨论中，若多边形的一个顶点的内角范围为 $0° < \theta < 180°$，则该顶点定义为凸顶点；否则，则定义为凹顶点。像前一段中指出的那样，当我们顺时针方向行进时，凸性是根据内部区域来度量的。

查找 MPP 的算法

属性1到4是查找一个MPP的顶点的基础。查找MPP的顶点的方法有多种（见Sklansky等[1972]以及 Kim 和 Sklansky[1982]）。我们在此处采用的方法是充分利用两个基本的 IPT/MATLAB 函数。第一个函数是qtdecomp，该函数执行四叉树分解，从而形成一堵包围感兴趣数据的细胞墙。第二个是函数 inpolygon，该函数用于确定哪些点位于由给定顶点集定义的多边形边界的外部、边界上或边界内部。

在一个示例中开发查找 MPP 的过程是很用帮助的。为此，我们再次使用图 11.3 和图 11.4。查找图11.4(a)中阴影内部区域的4连接边界的一种方法将在本节稍后讨论。在得到边界后，下一步就是寻找其角点，我们可通过获得其Freeman链码来这样做。代码方向的改变表示边界中的一个角点。沿着边界顺时针行进，通过分析方向的变化，很容易确定和标识出凸角和凹角，如图 11.4(b)所示。具体获得标识符的方法总结于本节稍后讨论的 M 函数minperpoly中。由这种方法确定的角点如图11.4(b)所示，我们在图11.5(a)中再一次显示了它们。为便于参照，我们给出了阴影区域和背景网格。为避免与多边形区域混淆，图 11.5 中显示的多边形未显示阴影区域的边界。

接下来，我们只使用初始的凸顶点（黑点）来构建一个初始多边形，如图 11.5(b)所示。由属性2可知，MPP的凸顶点集合是这个初始凸顶点集合的一个子集。可以看到，所有位于初始多边形外部的凹顶点（白点）不在多边形内形成凹性。对于在后续算法中变成凸性的那些特殊顶点，多边形将不得不穿过这些顶点。但我们知道，这些顶点永远不会变成凸性顶点，因为所有可能呈凸性的顶点均在此处确定（它们的角度在以后可能会变为 180°，但这对我边形的形状无影响）。因此，通过进一点的分析，可消除初始多边形外部的白点，如图 11.5(c)所示。

多边形内部的凹顶点（白点）与第一次通过时忽略的边界中的凹性有关。因此，这些顶点必须合并到多边形中，如图 11.5(d)所示。这时，通常会存在一些为黑点的顶点，但它们在新的多边形中不再是凸顶点［见图 11.5(d)中用箭头标示的黑点］。以上现象有两种可能的原因。第一种原因是这些顶点是图 11.5(b)中初始多边形的一部分，包含所有的凸（黑）顶点。第二种原因是合并一些额外的（白）顶点到多边形中会使得这些顶点变成凸顶点，如图 11.5(d)所示。因此，多边形中所有的黑点必须经过测试，以便查看在这些点处的顶点角是否超过了 180°。顶点角超过了 180° 的点必须删除。重复这一过程，直到删除了所有这些点。

图 11.5(e)中仅显示了一个新的黑顶点，在第二次通过该数据时该顶点变成了凹顶点。当没有顶点变化发生时，整个过程结束。此时所有角度为 180° 的顶点均会删除，因为它们不再位于边缘上，从而不会影响最终多边形的形状。图 11.5(f)中的边界就是我们的例子中的 MPP。该多边形与图 11.3(b)中的多边形相同。最后，图 11.4(g)显示了叠加在该 MPP 上的原始细胞联合体。

上述讨论可总结为查找一个区域的 MPP 的如下步骤：

1. 获取细胞联合体（该方法将在本节稍后讨论）。
2. 获取细胞联合体的内部区域。
3. 使用函数 boundaries 以 4 连接顺时针坐标序列的形式获得步骤 2 中的区域的边界。
4. 使用函数 fchcode 获得该 4 连接序列的 Freeman 链码。
5. 从链码中获得凸顶点（黑点）与凹顶点（白顶点）。
6. 使用黑点作为顶点构造一个初始多边形，在进一步的分析中删除位于该多边形之外的任何白顶点（在多边形边界上的白顶点将保留）。

7. 用剩余的黑白点作为顶点构造一个多边形。
8. 删除所有为凹顶点的黑点。
9. 重复步骤 7 与步骤 8，直到变化停止。此时，所有角度为 180° 的顶点均将删除。剩下的点就是该 MPP 的顶点。

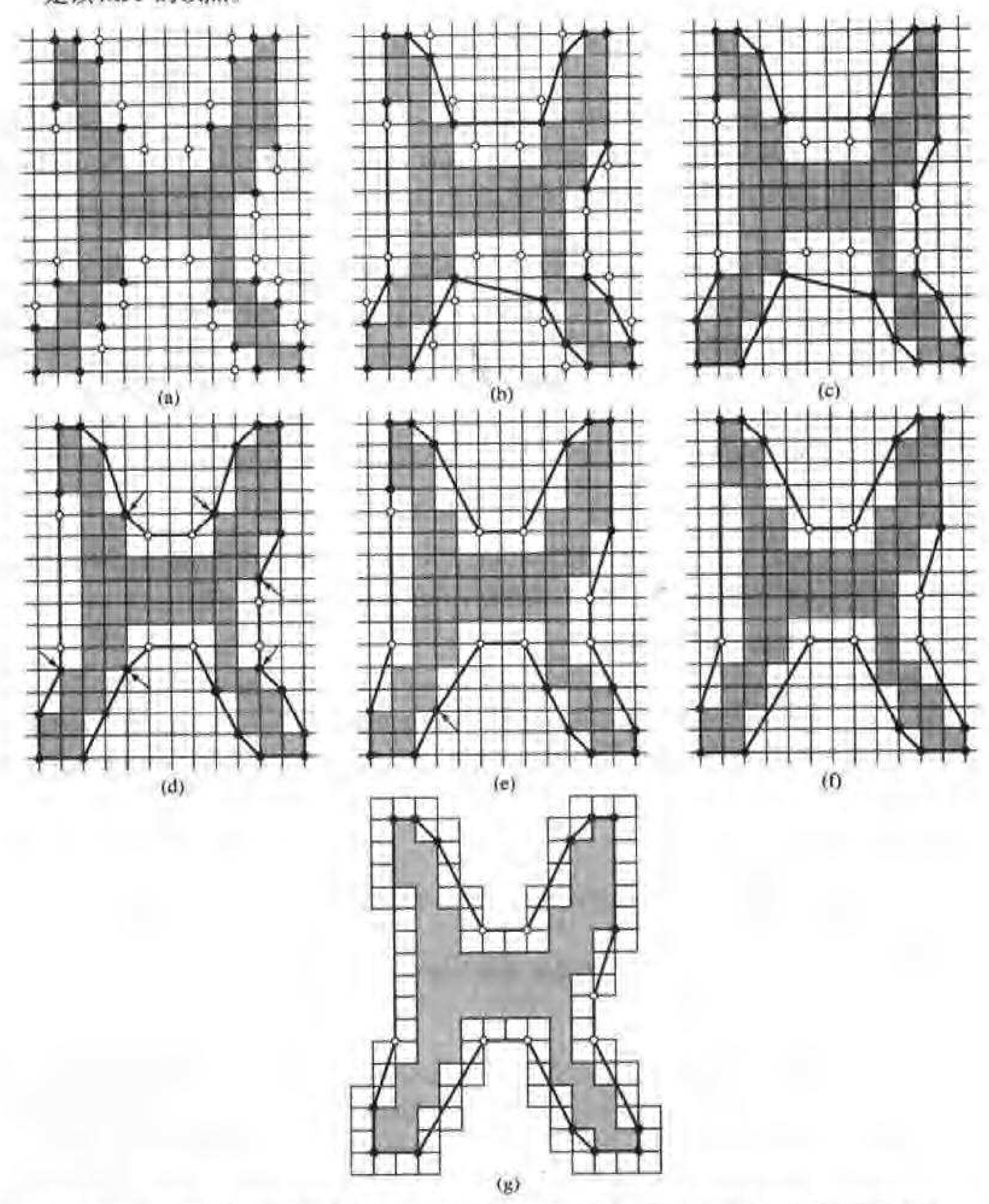


图 11.5　(a)图 11.4(a)中边界的凸（黑）顶点与凹（白）顶点；(b)连接所有凸顶点的初始多边形；(c)删除边界外部的凹顶点后的结果；(d)剩余凹顶点形成多边形后的结果（箭头表示已变为凹顶点且将被删除的黑点）；(e)删除凹（黑）顶点后的结果（箭头表示现在已变为凹顶点的黑顶点）；(f)显示 MPP 的最终结果；(g)叠加边界元素后的 MPP

MPP 算法实现中用到的一些 M 函数

我们可使用在10.4.2节中介绍的函数qtdecomp作为获得包围边界的细胞联合体的第一步。通常，我们考虑区域B，它由1和背景0的组成。函数qtdecomp在此处的语法为

```
Q = qtdecomp(B, threshold, [mindim maxdim])
```

其中，Q是一个包含四叉树结构的稀疏矩阵。若Q(k,m)非零，则(k,m)为分解中的左上块，块的大小为Q(k,m)。

若块元素的最大值减去块元素的最小值大于threshold，则分裂块。无论输入图像是何类图像，这个参数的值在0和1之间。使用前面的语法，函数qtdecomp将不会产生小于mindim或者大于maxdim的块。即使不满足以上的阈值条件，大于maxdim的块也将被分裂。maxdim/mindim的比值必须是2的幂。

若仅指定了以上两个值中的一个（无方括号），则函数假设它为mindim。这是我们在本节中使用的表述。图像B的大小必须是K × K，以便比值K/mindim是2的整数次幂。显然，K的最小可能值是B的最大维数。通常，B的大小需要通过在函数padarray中使用选项'post'于B中添加0来满足。例如，假设B的大小为640 × 480像素，且mindim = 3，则参数K必须满足K >= max(size(B))和K/mindim = 2^p，或K = mindim*(2^p)。对p求解，得p = 8，此时K = 768。

为获得四叉树分解中块的值，我们可使用在10.4.2节中讨论过的函数qtgetblk：

```
[vals, r, c] = qtgetblk(B, Q, mindim)
```

其中，vals是一个在B的四叉树分解中包含mindim × mindim个块的值的数组，而Q是由函数qtdecomp返回的稀疏矩阵，参数r和c是包含左上角的行和列坐标的向量。

例 11.4 获得一个区域的边界的细胞墙

为了解MPP算法中的步骤1到步骤4是如何实现的，可考虑图11.6(a)所示的图像，同时假设mindim = 2。我们将单个像素显示为小方形，以方便对函数qtdecomp的解释。图像大小为32 × 32，且很容易验证mindim的指定值不需要额外的填充。区域的4连接边界可使用如下命令得到：

```
>> B = bwperim(B, 8);
```

图11.6(b)显示了结果。注意，B仍然是一幅图像，此时它只包含一个4连接边界（记住，小方形对应的是单个像素）。

图11.6(c)显示了B的四叉树分解，它是使用如下命令得到的：

```
>> Q = qtdecomp(B, 0, 2);
```

其中，0用做阈值，以便块分裂成最小的2 × 2尺寸，而不管它们包含的1和0是如何混合的（每个这样的块能包含0到4个像素）。注意，许多块的尺寸大于2 × 2，但它们是相似的。

接着，我们使用了qtgetblk(B,Q,2)来提取值和大小为2 × 2的所有块的左上角坐标。然后，至少包含值为1的一个像素的所有块都将被1填充。结果用BF表示，如图11.6(d)所示。图像中的暗单元构成了细胞联合体。换言之，这些单元形成了图11.6(b)中的边界。

图11.6(d)中被细胞联合体围绕的区域可使用如下命令获得：

```
>> R = imfill(BF, 'holes') & ~BF;
```

图 11.6(e)显示了结果。若我们对这一区域的 4 连接边界感兴趣，则可使用如下命令来获得该边界：

```
>> b = boundaries(b, 4, 'cw');
>> b = b{1};
```

图 11.6(f)显示了结果。该图显示的 Freeman 链码可以使用函数 fchcode 获得。由此完成了算法 MPP 的步骤 1 到步骤 4。

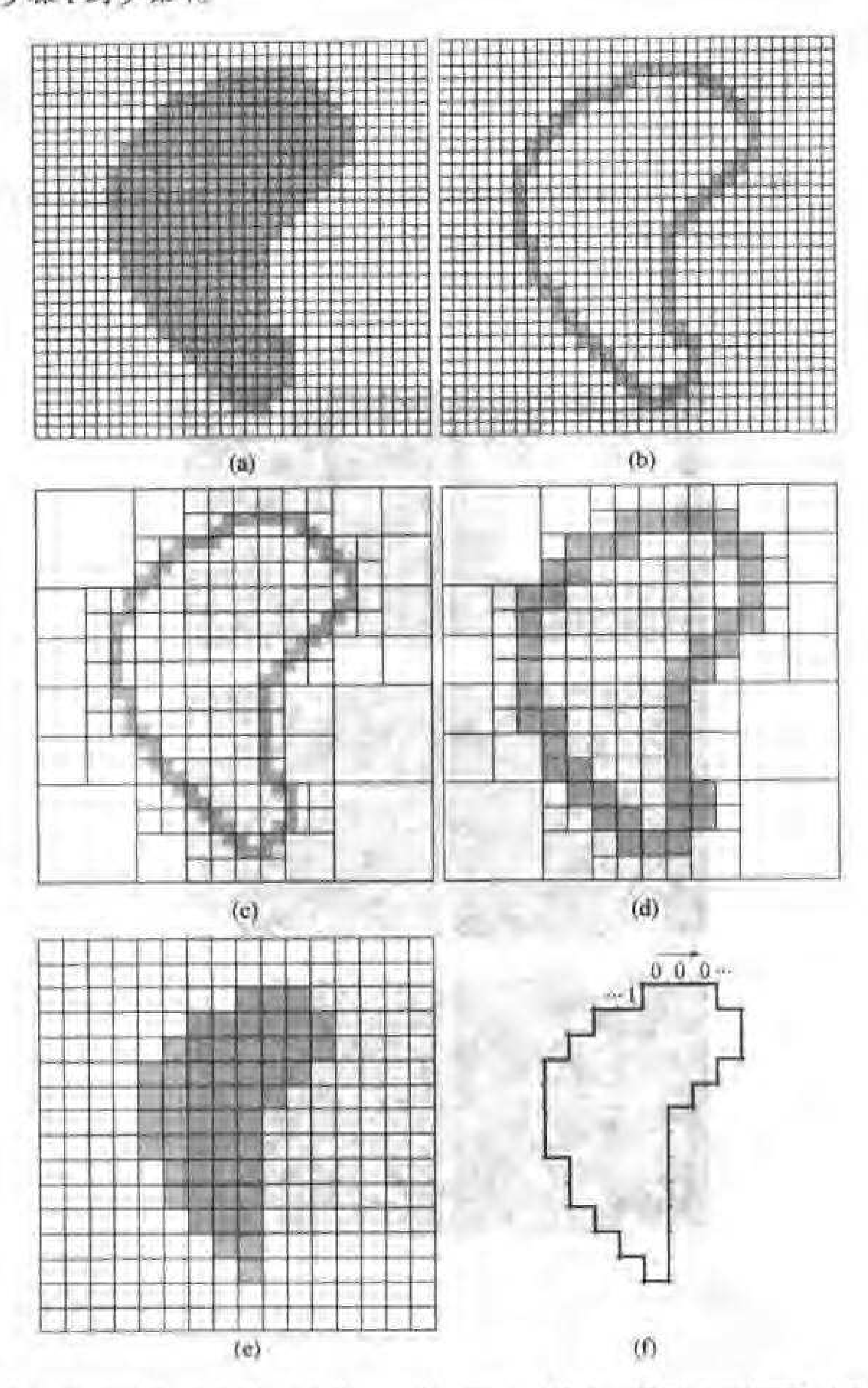


图 11.6　(a)原图像，其中的小方形表示单个像素；(b)4 连接边界；(c)使用最小尺寸为 2 像素的块的四叉树分解；(d)使用 1 填充所有大小为 2 × 2 的块后的结果，该块至少包含一个值为 1 的元素。这是细胞联合体；(e)图(d)的内部区域；(f)使用函数 boundaries 得到的 4 连接边界点。链码是使用函数 fchcode 得到的

函数inpolygon用在函数minperpoly（在下一节中讨论）中，以便决定一个点是否在多边形的外部、边界上或者多边形的内部，其语法为

```
IN = inpolygon(X, Y, xv, yv)
```

其中，X和Y是包含待测点的x和y坐标的向量，而xv和yv是包含按顺时针或逆时针顺序安排的多边形顶点的x和y坐标的向量。数组IN是一个向量，其长度等于待测点数。若点在多边形边界的内部或边界上，则其值为1；若点在边界外部，则其值为0。

计算MPP的M函数

MPP算法的步骤1到步骤9在函数minperpoly中实现，其程序清单包含在附录C中。语法为

```
[x, y] = minperpoly(B, cellsize)
```

其中，B是一幅输入二值图像，它包含单个区域或边界，而cellsize是细胞联合体中用于形成边界的方形单元的大小。列向量x和y包含MPP顶点的x坐标和y坐标。

例11.5　使用函数minperpoly

图11.7(a)显示了一幅枫叶图像B，图11.7(b)是使用如下命令获得的边界：

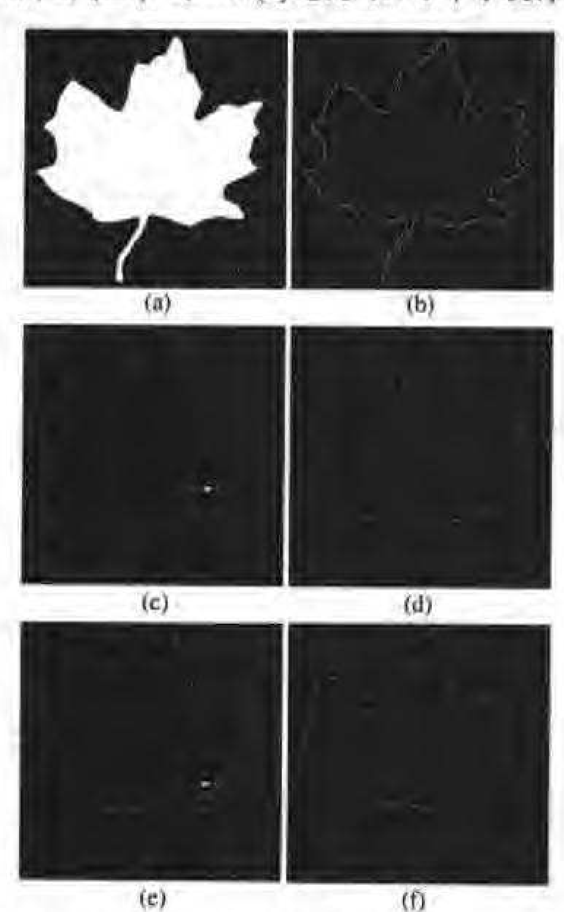


图11.7　(a)原图像；(b)4连接边界；(c)使用大小为2的方形边界单元获得的MPP；(d)~(f) 分别使用大小为3、4和8的方形单元获得的MPP

```
>> b = boundaries(B, 4, 'cw');
>> b = b{1};
>> [M, N] = size(B);
>> xmin = min(b(:, 1));
```

(X,Y) 与我们的图像坐标 (x,y) 之间的关系为 X = y，Y = -x [见图 2.1(a)]。函数 pol2cart 用于将极坐标转换为笛卡儿坐标：

```
[X, Y] = pol2cart(THETA, RHO)
```

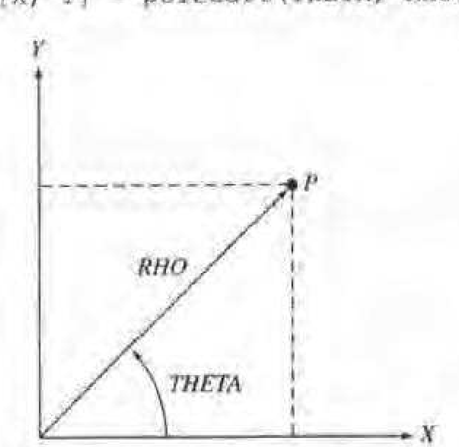


图 11.10　MATLAB 用于执行极坐标和笛卡儿坐标（以及笛卡儿坐标和极坐标）转换的坐标轴约定

例 11.6　标记

图 11.11(a)和图 11.11(b)分别显示了嵌入在大小为 674 × 674 像素的矩阵中的不规则方形边界 bs 和三角形边界 bt。图 11.11(c)显示了使用如下命令得到的方形标记：

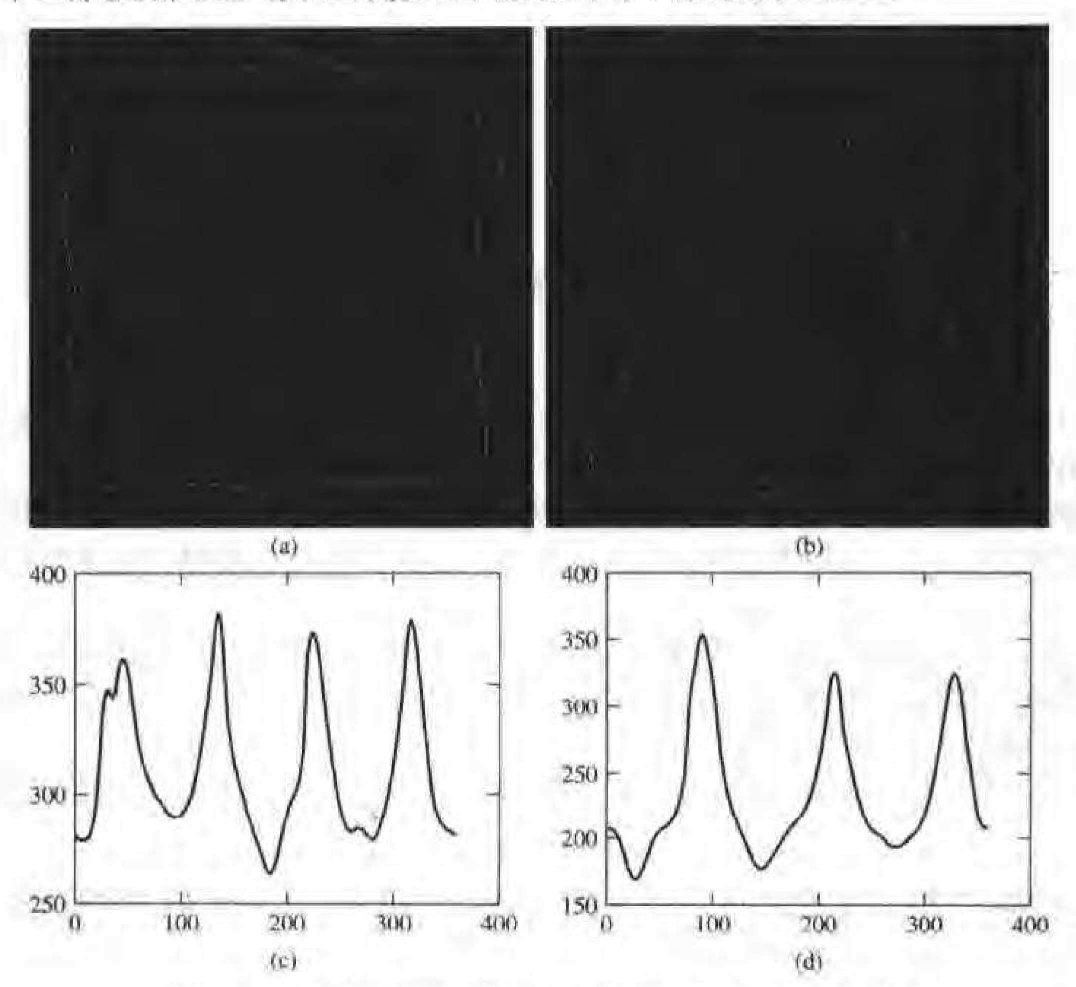


图 11.11　(a)和(b)不规则方形与三角形边界；(c)和(d)相应的标记

```
>> [st, angle, x0 , y0 ] = signature(bs);
>> plot(angle, st)
```

用以上命令得到的 x0 和 y0 的值是[342, 326]。一对类似的命令产生了图 11.11(d)所示的图形，其质心位于[416, 335]。注意，简单地计算两个标记中的显著峰值数足以区分这两个边界。

11.2.4 边界片断

将一条边界分解为片断通常是很有用的。分解降低了边界的复杂度，从而简化了描述过程。当边界包含一个或者多个携带形状信息的重要凹面时，这种方法尤其具有吸引力。在这种情况下，使用由边界包围的区域凸壳对边界的鲁棒分解是一种有力的工具。

任意集合 S 的凸壳 H 是包含 S 的最小凸集。集合的差 H-S 称为 S 的凸缺 D。为了解如何使用这些概念将边界分成有意义的片断，考虑图 11.12(a)，其显示了一个对象（集合 S）及其凸缺（阴影区域）。区域边界可以通过沿 S 轮廓线标出进入或者离开凸缺组成部分时引起变化的点来加以分割。图 11.12(b)显示了这种情况下的结果。原理上，该方案独立于区域大小和方向。在实际中，这种处理要先于减少“非重要”凹面数目的图像平滑处理完成。在刚刚讨论的方法中，执行边界分解的必要 MATLAB 工具包含在函数 regionprops 中，该函数将在 11.4.1 节中讨论。

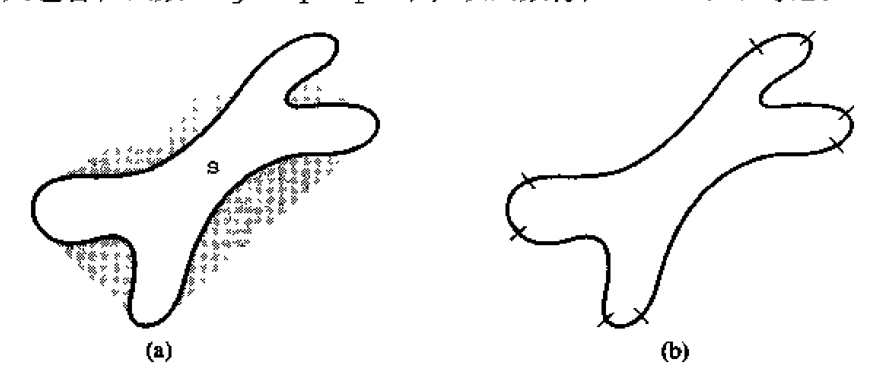


图 11.12 (a)区域 S 及其凸缺（阴影）；(b)分割后的边界

11.2.5 骨骼

用于表示平面区域结构形状的一种重要方法是将其简化为一幅图形。这一简化可以通过一种细化（也称为骨骼化）算法获取区域骨骼来完成。

区域骨骼可以通过中间轴变换（MAT）加以定义。一个边概为 b 的区域 R 的 MAT 描述如下：对于 R 中的每个点 p，寻找 b 中的最近邻点。若 p 比这样的近邻点大，则我们称 p 属于 R 的中间轴线（骨骼）。

虽然一个区域的 MAT 是一个直观的概念，但该定义的直接实现在计算上是昂贵的，因为它涉及到计算每个内部点到区域的边界上每个点的距离。人们已提出了许多算法，用以改进计算的效率，同时试图近似表示区域的中间轴线。

如 9.3.4 节所述，IPT 通过函数 bwmorph 使用如下语法来生成二值图像 B 中所有区域的骨骼：

```
S = bwmorph(B , 'skel', Inf)
```

该函数会删除对象的边界上的像素，但不允许对象断开。剩下的像素则构成了图像的骨骼。这一选择保存了（定义于表 11.1 中的）欧拉数。

例 11.7　计算一个区域的骨骼

图 11.13(a)所示的图像 f 是人类染色体经电子显微镜放大 30 000 倍且经分割后的图像。本例的目的是计算染色体的骨骼。

显然，处理的第一步是将染色体与背景无关的细节加以分离。一种方法是对图像进行平滑处理，然后对其进行阈值处理。图 11.13(b)显示了使用一个 25 × 25 高斯空间掩模（sig = 15）平滑图像 f 后的结果：

```
>> f = im2double(f);
>> h = fspecial('gaussian', 25, 15);
>> g = imfilter(f, h, 'replicate');
>> imshow(g) % Fig. 11.13(b)
```

接下来，我们对平滑后的图像进行阈值处理：

```
>> g = im2bw(g, 1.5*graythresh(g));
>> figure, imshow(g) % Fig. 11.13(c)
```

其中，自动确定的阈值 graythresh(g) 已乘 1.5，以增加 50% 的阈值处理量。这样做的原因在于增加阈值可增加从边界中删除的数据数量，从而可实现进一步的平滑。图 11.13(d)所示的骨骼是使用如下命令得到的：

```
>> s = bwmorph(g, 'skel', Inf); % Fig. 11.13(d)
```

骨骼中的毛刺可通过如下命令删除：

```
>> s1 = bwmorph(s, 'spur', 8); % Fig. 11.13(e)
```

其中，我们已重复该操作 8 次，这种情况下近似等于平滑滤波器中 sig 值的一半。一些小毛刺仍然存在于骨骼中。但再应用该函数 7 次（完成 sig 的值）产生了如图 11.13(f)所示的结果，该结果是输入的合理骨骼表示。经验表明，高斯平滑掩模的 sig 值可用于确定应用去除毛刺算法的次数。

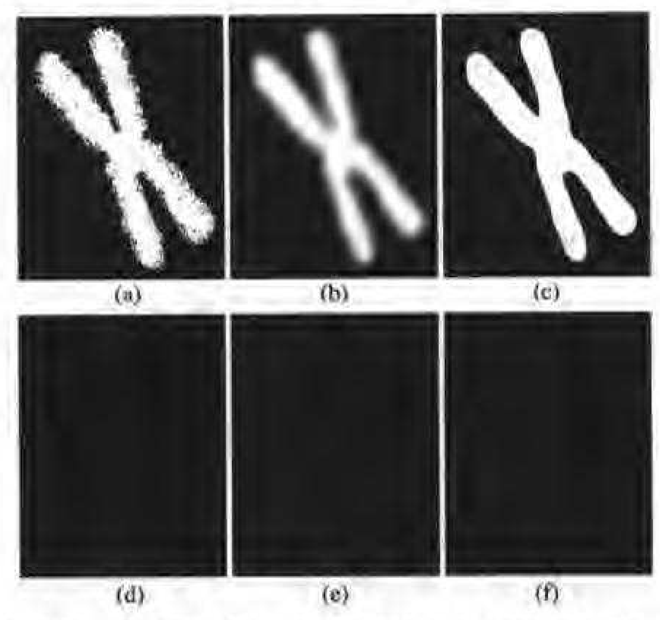


图 11.13　(a)分割后的人类染色体；(b)使用 25 × 25 高斯空间掩模（sig = 15）平滑图像 f 后的结果；(c)经阈值处理后的图像；(d)骨骼；(e)应用 8 次去除毛刺的算法后的骨骼；(f)再应用 7 次去除刺算法后的骨骼

数可由函数 fchcode 的参数 c.diffmm 给出，该函数在 11.2.1 节中讨论过，并且形状数的阶计算为 length（c.diffmm）。

正如 11.2.1 节中所述的那样，4 方向 Freeman 链码可通过使用最小幅度的整数，使其对起始点不敏感，并可通过使用码的一阶差分，使其对于 90° 的倍数的旋转不敏感。这样，形状数就对起点以及 90° 倍数的旋转均不敏感。频繁用于归一化任意旋转的一种方法是使坐标轴这一与长轴对齐，然后基于旋转后图形提取 4 方向链码。图 11.14 示例了该过程。

目前，人们已开发了许多用于计算形状数的 M 函数，包括提取边界的函数 boundaries、查找长轴的函数 diameter、降低取样网格分辨率的函数 bsubsamp 和提取形状数的函数 fchcode 组成。记住，在使用函数 boundaries 提取 4 方向链码边界时，输入图像必须是用 bwlabel 指定的 4 连通性图像。如图 11.14 中指出的那样，旋转的补偿是以坐标轴与长轴方向一致为基础的。函数 x2majoraxis 可使 x 轴与区域或边界的长轴一致，该函数的语法为（其代码包含在附录 C 中）

```
[B, theta] = x2majoraxis(A, B)
```

其中，A = s.MajorAxis 源于函数 diameter，B 是一幅输入二值图像或边界列表（与前面一样，假设边界是已连接的封闭曲线）。输出 B 与输入具有相同的形式（如一幅二值图像或一个坐标序列）。因为可能的舍入误差，旋转可能会产生一个分离的边界序列，所以可能需要进行后处理（使用函数 bwmorph），以便重新将这些点连接起来。

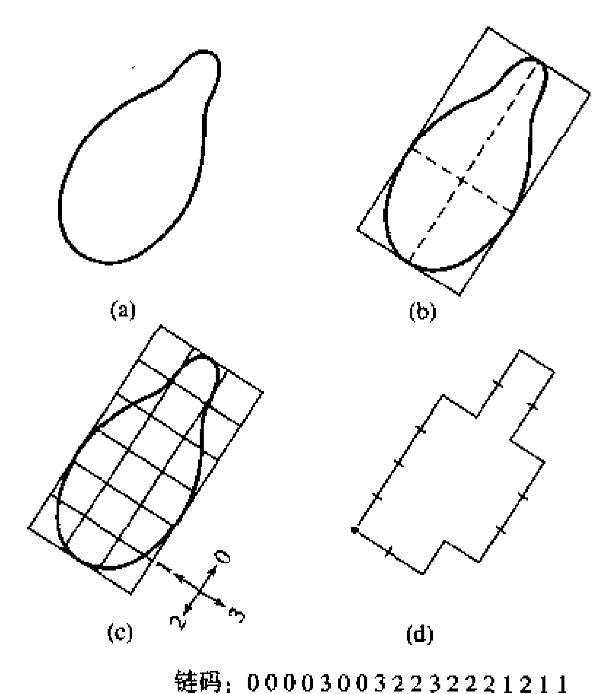


图 11.14　生成形状数的步骤

11.3.3　傅里叶描绘子

图 11.15 显示了 xy 平面中的 K 点数字边界。在任意点(x_0, y_0)开始，沿边界逆时针方向行进会出现坐标对$(x_0, y_0), (x_1, y_1), (x_2, y_2), \cdots, (x_{K-1}, y_{K-1})$。这些坐标可以表示为 $x(k) = x_k$, $y(k) = y_k$。使用这种表达方法，边界本身可以表示为坐标序列 $s(k) = [x(k), y(k)]$, $k = 0, 1, 2, \cdots, K-1$。进而，每个坐标对可当做一个复数来处理，从面得到

$$s(k) = x(k) + jy(k)$$

从4.1节可知，$s(k)$的离散傅里叶变换（DFT）为

$$a(u) = \sum_{k=0}^{K-1} s(k)\mathrm{e}^{-j2\pi uk/K}$$

其中，$u = 0, 1, 2, \cdots, K-1$。复系数$a(u)$称为边界的傅里叶描绘子。这些系数的傅里叶逆变换可恢复$s(k)$，即

$$s(k) = \frac{1}{K}\sum_{u=0}^{K-1} a(u)\mathrm{e}^{j2\pi uk/K}$$

其中，$k = 0, 1, 2, \cdots, K-1$。假设我们仅使用前P个傅里叶系数，而不使用所有系数。这相当于在上式中令$a(u) = 0, u > P-1$。结果是$s(k)$的如下近似：

$$\hat{s}(k) = \frac{1}{P}\sum_{u=0}^{P-1} a(u)\mathrm{e}^{j2\pi uk/K}$$

其中，$k = 0, 1, 2, \cdots, K-1$。虽然仅使用P个傅里叶系数可得到$\hat{s}(k)$的每一个分量，但k的范围仍是从0到$K-1$。换言之，近似边界中包含有相同数量的点，但在每个点的重构中却用不到如此多的系数项。回忆第4章可知，高频分量决定细节部分，低频分量决定总体形状。因此，随着P的减少，边界细节的损失就会增加。

下面的函数 frdescp 用于计算边界的傅里叶描绘子 s。类似地，给定一组傅里叶描绘子，函数 ifrdescp 可用给定数量的描绘子计算其逆变换，以产生一条封闭的空间曲线。每个函数的文本段均解释了其语法。

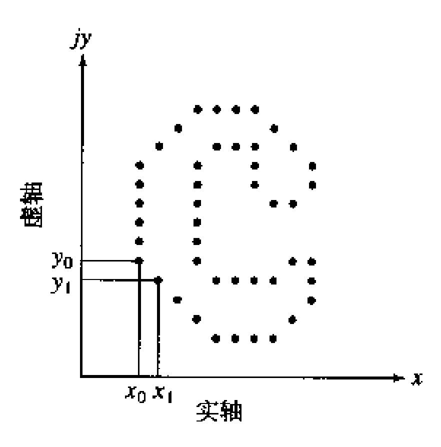


图 11.15　一个数字边界及其复数序列表示。点(x_0, y_0)和(x_1, y_1)是序列中的前两个点

```
function z = frdescp(s)
%FRDESCP Computes Fourier descriptors.
%   Z = FRDESCP(S) computes the Fourier descriptors of S, which is an
%   np-by-2 sequence of image coordinates describing a boundary.
%
%   Due to symmetry considerations when working with inverse Fourier
%   descriptors based on fewer than np terms, the number of
%   points in S when computing the descriptors must be even. If the
%   number of points is odd, FRDESCP duplicates the end point and
%   adds it at the end of the sequence. If a different treatment is
%   desired, the sequence must be processed externally so that it has
%   an even number of points.
```

```
%
%   See function IFRDESCP for computing the inverse descriptors.

% Preliminaries
[np, nc] = size(s);
if nc ~= 2
   error('S must be of size np-by-2.');
end
if np/2 ~= round(np/2);
   s(end + 1, :) = s(end, :);
   np = np + 1;
end

% Create an alternating sequence of 1s and -1s for use in centering
% the transform.
x = 0:(np - 1);
m = ((-1) .^ x)';

% Multiply the input sequence by alternating 1s and -1s to
% center the transform.
s(:, 1) = m .* s(:, 1);
s(:, 2) = m .* s(:, 2);
% Convert coordinates to complex numbers.
s = s(:, 1) + i*s(:, 2);
% Compute the descriptors.
z = fft(s);
```

函数ifrdescp 如下所示:

```
function s = ifrdescp(z, nd)
%IFRDESCP Computes inverse Fourier descriptors.
%   I = IFRDESCP(Z, ND) computes the inverse Fourier descriptors of
%   of Z, which is a sequence of Fourier descriptor obtained, for
%   example, by using function FRDESCP. ND is the number of
%   descriptors used to computing the inverse; ND must be an even
%   integer no greater than length(Z). If ND is omitted, it defaults
%   to length(Z). The output, S, is an ND-by-2 matrix containing the
%   coordinates of a closed boundary.

% Preliminaries.
np = length(z);
% Check inputs.
if nargin == 1 | nd > np
   nd = np;
end

% Create an alternating sequence of 1s and -1s for use in centering
% the transform.
x = 0:(np - 1);
m = ((-1) .^ x)';

% Use only nd descriptors in the inverse. Since the
% descriptors are centered, (np - nd)/2 terms from each end of
% the sequence are set to 0.
d = round((np - nd)/2); % Round in case nd is odd.
z(1:d) = 0;
z(np - d + 1:np) = 0;
% Compute the inverse and convert back to coordinates.
```

```
zz = ifft(z);
s(:, 1) = real(zz);
s(:, 2) = imag(zz);
% Multiply by alternating 1 and -1s to undo the earlier
% centering.
s(:, 1) = m.*s(:, 1);
s(:, 2) = m.*s(:, 2);
```

例 11.8 傅里叶描绘子

图 11.16(a)显示了一幅二值图像 f，该图像类似于图 11.13(c)所示的图像，但是是使用 sigma = 9 的 15 × 15 高斯掩模且阈值为 0.7 获得的。其目的是产生一幅并不完全光滑的图像，以便用它来说明减少描绘子的数量对边界形状产生的影响。图 11.16(b)所示的图像是使用如下命令生成的：

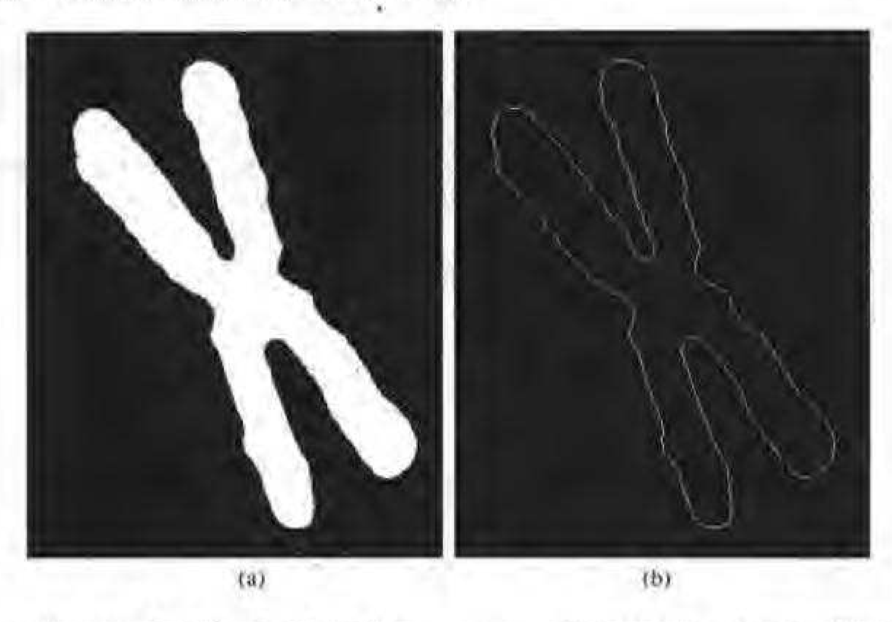
```
>> b = boundaries(f);
>> b = b{1};
>> bim = bound2im(b, 344, 270);
```

其中显示的维数是图像 f 的维数。图 11.16(b)显示了图像 bim。显示的边界有 1090 个点。下面，我们计算傅立叶描绘子，

```
>> z = frdescp(b);
```

且使用约 1090 个描绘子中的 50% 获得其逆变换：

```
>> z546 = ifrdescp(z, 546);
>> z546im = bound2im(z546, 344, 270);
```



(a) (b)

图 11.16 (a)二值图像；(b)使用函数 boundaries 提取的边界。边界有 1090 个点

图像 z546im［见图 11.17(a)］与图 11.16(b)中的原始边界几乎完全一致。一些细节信息已丢失，但从实用目的来讲，这两个边界是一样的。图 11.17(b)到图 11.17(f)分别显示了使用 110, 56、28、14、8 个描绘子重构的图像，这些描绘子的个数分别相当于原有 1090 个描绘子的 10%, 5%、2.5%、1.25% 和 0.7%。使用 110 个描绘子所产生的图像［见图 11.17(c)］显示了稍微平滑一些的边界，但产生的形状与原图像十分接近。图 11.17(e)显示了用原有描绘子总数的 1.25%

即 14 个描绘子重构的图像，它仅保留了边界的主要特征。图 11.17(f)因为丢失了边界的主要特征（4 个长的凸出部分）而显示了无法接受的失真。进一步减少到 4 个和 2 个描绘子时，所产生的图像为一个椭圆或圆。

由于像素值的舍入，图 11.17 中的某些边界有一个像素的缺口。这些小缺口是傅里叶描绘子共有的，可以用带 'bridge' 选项的函数 bwmorph 加以修复。

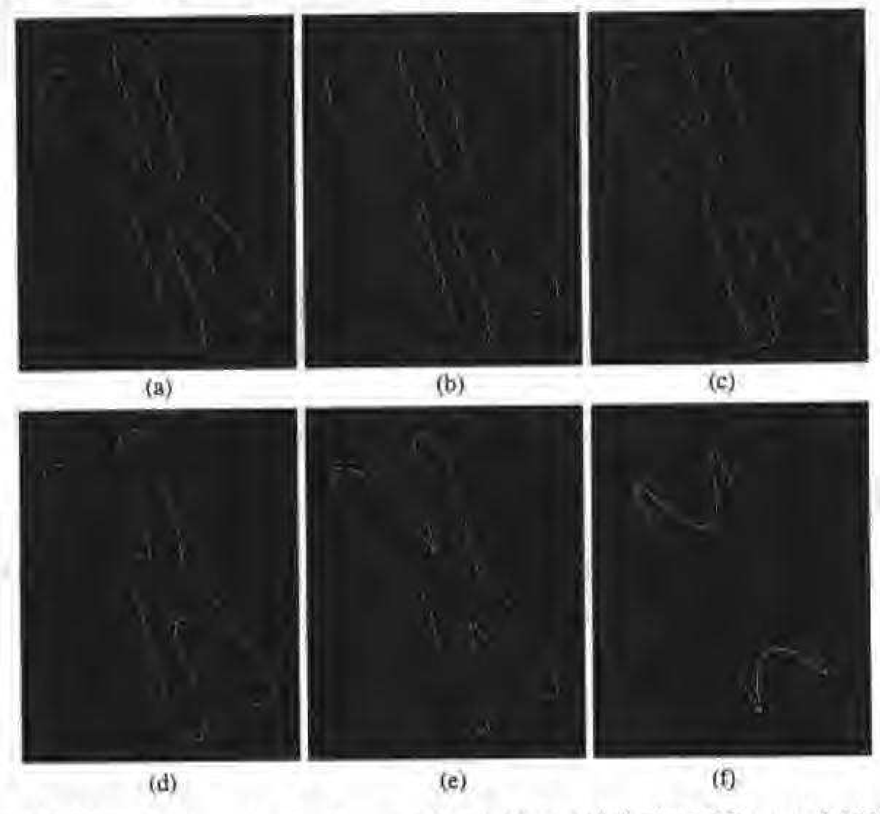


图 11.17 (a)~(f)分别使用 546，110，56，28，14 和 8 个傅里叶描绘子而不是 1090 个描绘子重构的边界

如前所述，描绘子应该尽可能地对平移、旋转和缩放等改变不敏感。当结果取决于所处理的点的顺序时，要给它们加一个额外的约束，以便使描绘子对起始点不敏感。傅里叶描绘子虽然对几何变化间接不敏感，但这些参数的变化却与描绘子的简单变换有关（见 Gonzalez and Woods[2002]）。

11.3.4 统计矩

一维边界表示的形状（如边界线段和标记波形）可以使用统计矩（如均值、方差和高阶矩）定量地描述。考虑图 11.18(a)和图 11.18(b)，图 11.18(a)显示了一个边界线段，图 11.18(b)显示了一个以任意变量 r 的一维函数 $g(r)$ 描绘的线段。该函数的获得方式如下：连接该线段的两个端点以形成一个主轴，然后使用 11.3.2 节中讨论的函数 x2majoraxis 来使长轴与 x 轴对齐。

描述 $g(r)$ 的形状的一种方法是将 $g(r)$ 归一化到单位面积内，并把它当做一个直方图来处理。换言之，$g(r_i)$ 可以看成是值 r_i 发生的概率。在这种情况下，r 可看做是一个随机变量，其矩为

$$\mu_n = \sum_{i=0}^{K-1} (r_i - m)^n g(r_i)$$

其中，

$$m = \sum_{i=0}^{K-1} r_i g(r_i)$$

（续表）

properties 的有效字符串	说明
'ConvexHull'	p × 2 矩阵；包含区域的最小凸多边形。矩阵的每一行包含多边形的 p 个顶点之一的 x 坐标和 y 坐标
'ConvexImage'	二值图像；凸壳，其中的所有像素均被填充（即置为 on）；对于凸壳边界通过的像素，regionprops 用与 roipoly 相同的逻辑来确决定该像素是在凸壳的内部还是在凸壳的外部。图像是区域的围框的大小
'Eccentricity'	标量；与区域有着相同二阶矩的椭圆的偏心率。偏心率是椭圆的焦距与主轴长度间距离的比率。其值在 0 和 1 之间，等于 0 和 1 是退化的情况（偏心率为 0 的椭圆是圆，偏心率为 1 的椭圆是线段）
'EquivDiameter'	标量；与区域有着相同面积的圆的直径。计算为 sqrt(4*Area/pi)
'EulerNumber'	标量；等于区域中的对象数减去这些对象中的孔洞数
'Extent'	标量；也在区域内的围框中的像素的比例。计算为围框的面积除 Area
'Extrema'	8 × 2 矩阵；区域中的极值点。矩阵的每一行包含一个点的 x 和 y 坐标。向量的格式为 [top-left, top-right, right-top, right-bottom, bottom-right, bottom-left, left-bottom, left-top]
'FilledArea'	FilledImage 中 on 像素的数目
'FilledImage'	大小与区域的围框相同的二值图像。on 像素对应于所有孔洞均已填充的区域
'Image'	大小与区域的围框相同的二值图像。on 像素对应于该区域，其他像素为 off 像素
'MajorAxisLength'	与区域有着相同二阶矩的椭圆的长轴 *** 的长度（像素数）
'MinorAxisLength'	与区域有着相同二阶矩的椭圆的短轴 *** 的长度（像素数）
'Orientation'	x 轴和与区域有着相同二阶矩的椭圆的长轴间的角度（单位为度）
'PixelList'	行为区域内实际像素的[x, y]坐标的矩阵
'Solidity'	标量；也在区域内的凸壳中的像素的比例。计算为 Area/ConvexArea

***注意，本文中使用的长轴和短轴不同于 11.3.1 节中讨论的基本矩形中的长轴和短轴。关于椭圆的矩的讨论，请参阅 Haralick 和 Shapiro[1992]。

例 11.9　使用函数 regionprops

作为一个简单的示例，假设我们想获得图像 B 中每个区域的面积和围框。我们写出如下命令：

```
>> B = bwlabel(B); % Convert B to a label matrix.
>> D = regionprops(B, 'area', 'boundingbox');
```

要提取面积和区域数，我们写出

```
>> w = [D.Area];
>> NR = length(w);
```

其中，向量 w 的元素是区域的面积，NR 是区域的个数。同样，我们可以获得单个矩阵，它的行是用如下语句得到的每一个区域的围框：

```
V = cat(1, D.BoundingBox);
```

这个数组的维数是 NR × 4。cat 操作符曾在 6.1.1 节中讨论过。

11.4.2　纹理

描述区域的一种重要方法是量化区域的纹理内容。在这一节中，我们将说明两个新函数的用法，以基于统计方法和谱度量方法来计算纹理。

统计方法

频繁用于纹理分析的一种方法是以亮度直方图的统计属性为基础的。一类这样的度量基于统计矩。正如 5.2.4 节中所讨论的那样，均值的第 n 阶矩表示为

$$\mu_n = \sum_{i=0}^{L-1}(z_i - m)^n p(z_i)$$

其中，z_i是表示亮度的一个随机变量，$p(z)$是一个区域中的灰度级的直方图，L是可能的灰度级数，而

$$m = \sum_{i=0}^{L-1} z_i p(z_i)$$

则是均值（平均）亮度。这些矩可使用5.2.4节中讨论的函数 statmoments 来计算。表11.2列出了基于统计矩、一致性和熵的常用描绘子。记住，二阶矩$\mu_2(z)$是方差σ^2。

表 11.2 基于区域的亮度直方图的某些纹理描绘子

矩	表达式	纹理度量
均值	$m = \sum_{i=0}^{L-1} z_i p(z_i)$	平均亮度度量
标准偏差	$\sigma = \sqrt{\mu_2(z)} = \sqrt{\sigma^2}$	平均对比度度量
平滑度	$R = 1 - 1/(1 + \sigma^2)$	区域中亮度的相对平滑度度量。对于常亮度区域，R等于0；对于灰度级的值有着较大偏移的区域，R等于1。实践中，该度量中使用的方差通过除以$(L-1)^2$，可归一化到范围[0, 1]
三阶矩	$\mu_3 = \sum_{i=0}^{L-1}(z_i - m)^3 p(z_i)$	度量直方图的偏斜。若直方图是对称的，则度量为0；若度量为正值，则直方图向右偏斜，若度量为负值，则直方图向左偏斜。该度量的值在某一范围内，与其他5种度量类似，使用相同的除数$(L-1)^2$除μ_3，我们可将方差归一化
一致性	$U = \sum_{i=0}^{L-1} p^2(z_i)$	度量一致性。当所有灰度值相等时，该度量最大并从此处开始减小
熵	$e = -\sum_{i=0}^{L-1} p(z_i)\log_2 p(z_i)$	随机性度量

编写一个M函数来计算表11.3中的纹理度量是很简单的。为此目的所写的函数statxture包含在附录C中。该函数的语法为

```
t = statxture(f, scale)
```

其中，f是一幅输入图像（或子图像），t是一个6元素行向量，其分量是表11.2中的描绘子，这些描绘子都以相同的顺序排列。参数 scale 也是一个6元素行向量，为达到缩放的目的，它的元素要与 t 的相应元素相乘。若省略，则 scale 的默认值都是 1。

表 11.3 图 11.19 所示区域的纹理度量

纹理	平均亮度	平均对比度	R	第3阶矩	一致性	熵
平滑	87.02	11.17	0.002	−0.011	0.028	5.367
粗糙	119.93	73.89	0.078	2.074	0.005	7.842
周期	98.48	33.50	0.017	0.557	0.014	6.517

例 11.10 统计纹理度量

图11.19中由白色方框所包围的三个区域从左到右分别为光滑纹理、粗糙纹理和周期纹理。使用 imhist 函数获得的这些区域的直方图如图11.20所示。对图11.19中的每一个子图像应用函数 statxture，可获得表11.3中的各项数据。这些结果通常与各自的子图像的纹理内容一致。例如，粗糙区域［见图11.9(b)］的熵比其他两个区域的熵高，这是因为该区域的像素值较其他两种区域的像素值的随机性要大。在这种情况下，其对比度和平均亮度也要高一些。另外，正如由R值和一致性度量所显示的那样，该区域的平滑性和一致性是最低的。粗糙区域的

直方图也表明在均值位置上最缺乏对称性，这一点在图 11.20(b)中显示得非常清楚，同时，第三阶矩的最大值也显示于表 11.3 中。

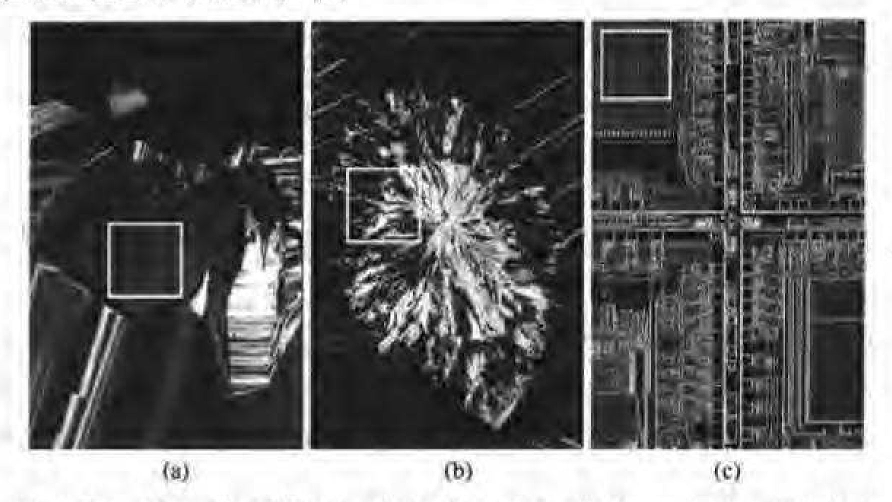


图 11.19　所显示的子图像从左到右分别表示平滑的、粗糙的、周期的纹理。它们分别是超导体、人类胆固醇和微处理器的光学显微图像（原图由佛罗里达州立大学的 Michael W. Davidson 博士提供）

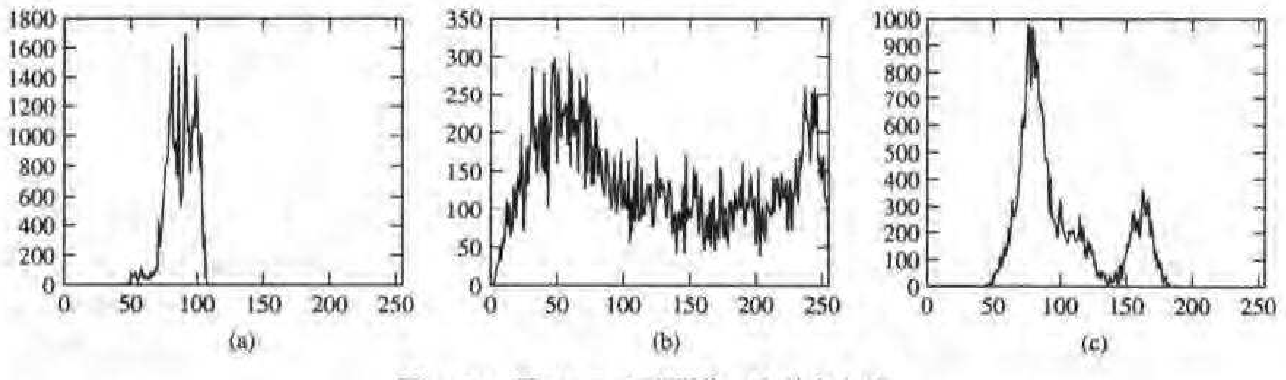


图 11.20　图 11.19 中子图像对应的直方图

纹理的频谱度量

纹理的频谱度量是基于傅里叶频谱的，适用于描述图像中的周期或近似周期二维模式的方向性。这些在频域中易于识别的全局纹理模式，在空间域中则很难检测到。因此，纹理的频谱对于判别周期纹理模式和非周期纹理模式非常有用，对于量化两个周期模式间的差也非常有用。

用极坐标表示频谱所得到的函数 $S(r,\theta)$ 可以简化频谱特性的解释，其中 S 是频谱函数，r 和 θ 是极坐标系中的变量。对于每一个方向 θ，我们可以将 $S(r,\theta)$ 看成是一个一维函数 $S_\theta(r)$ 同样，对于每一个频率 r，我们也可以将 $S_r(\theta)$ 看成是一个一维函数。θ 为定值时，分析 $S_\theta(r)$ 可以得到从原点出发沿半径方向的频谱特性；同理，r 为定值时，分析 $S_r(\theta)$ 可以得到以原点为中心的一个圆的频谱特性。

通过求这些函数的积分（离散变量求和），我们可得到全局描述：

$$S(r) = \sum_{\theta=0}^{\pi} S_\theta(r)$$

和

$$S(\theta) = \sum_{r=1}^{R_0} S_r(\theta)$$

其中，R_0 是中心在原点的圆的半径。

对每对坐标(r, θ)，这两个方程的结果构成了一个二元组$[S(r), S(\theta)]$。通过改变这些坐标，可以产生两个一维函数$S(r)$和$S(\theta)$，它们构成了整幅图像或所考虑区域的纹理的谱能量描述。此外，为了定量地表征它们的特性，这些函数自身的描绘子也是可计算的。用于该目的的典型描绘子为最大值的位置、振幅和轴向变化的均值与方差，以及函数最大值和均值之间的距离。

函数specxture（见附录C）可用于计算前面的两个纹理度量，其语法为

```
[srad, sang, S] = specxture(f)
```

其中，srad是$S(r)$，sang是$S(\theta)$，S是频谱图（使用第4章描述的函数log来显示）。

例11.11　计算谱纹理

图11.21(a)显示了一幅图像，图像中的对象的摆放是任意的，图11.21(b)显示了包含相同对象的图像，但这些对象已周期性地排列起来。使用函数specxture计算的相应傅里叶频谱图显示在图11.21(c)和图11.21(d)中。由于粗糙背景材质上火柴的摆放形成了周期纹理，所以在二维傅里叶频谱中出现了向四周扩展的周期突发。在图11.21(c)所示的频谱中，其他成分很明显是由图11.21(a)中任意排列的很强边缘产生的。通过对比可以看出，在图11.21(d)中，能量主要集中在水平轴方向，这些能量与背景无关，对应于图11.21(b)中的垂直强边缘。

图11.22(a)和图11.22(b)分别是火柴任意排列时$S(r)$和$S(\theta)$的图形，图11.22(c)和图11.22(d)分别是火柴整齐排列时$S(r)$和$S(\theta)$的图形，它们均是使用函数specxture计算的。这些图形是使用命令plot(srad)和plot(sang)得到的。图11.22(a)和图11.22(c)中的坐标轴的刻度是使用如下函数确定的：

```
>> axis([horzmin horzmax vertmin vertmax])
```

对应于火柴随机排列的$S(r)$的曲线无很强的周期分量（即在频谱中除了直流分量的起始位置有一个峰值外，没有其他的峰值）。另一方面，对应于火柴顺序排列的$S(r)$的曲线表明在$r = 15$处有一个强峰值，在$r = 25$处有一个较小的峰值。类似地，图11.21(c)中能量突发的随机性在图11.22(b)所示的$S(\theta)$曲线上也很明显。比较而言，图11.22(d)所示的曲线表明在原点附近的区域中以及在$\theta = 90^\circ$和$\theta = 180^\circ$处有较强的能量分量。这与图11.21(d)中的能量分布一致。

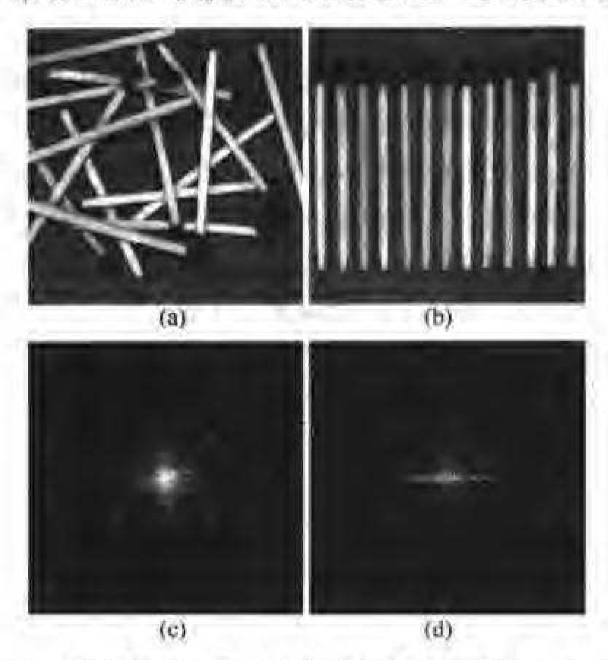


图11.21　(a)和(b)顺序排列和非顺序排列的对象的图像；(c)和(d)相应的频谱

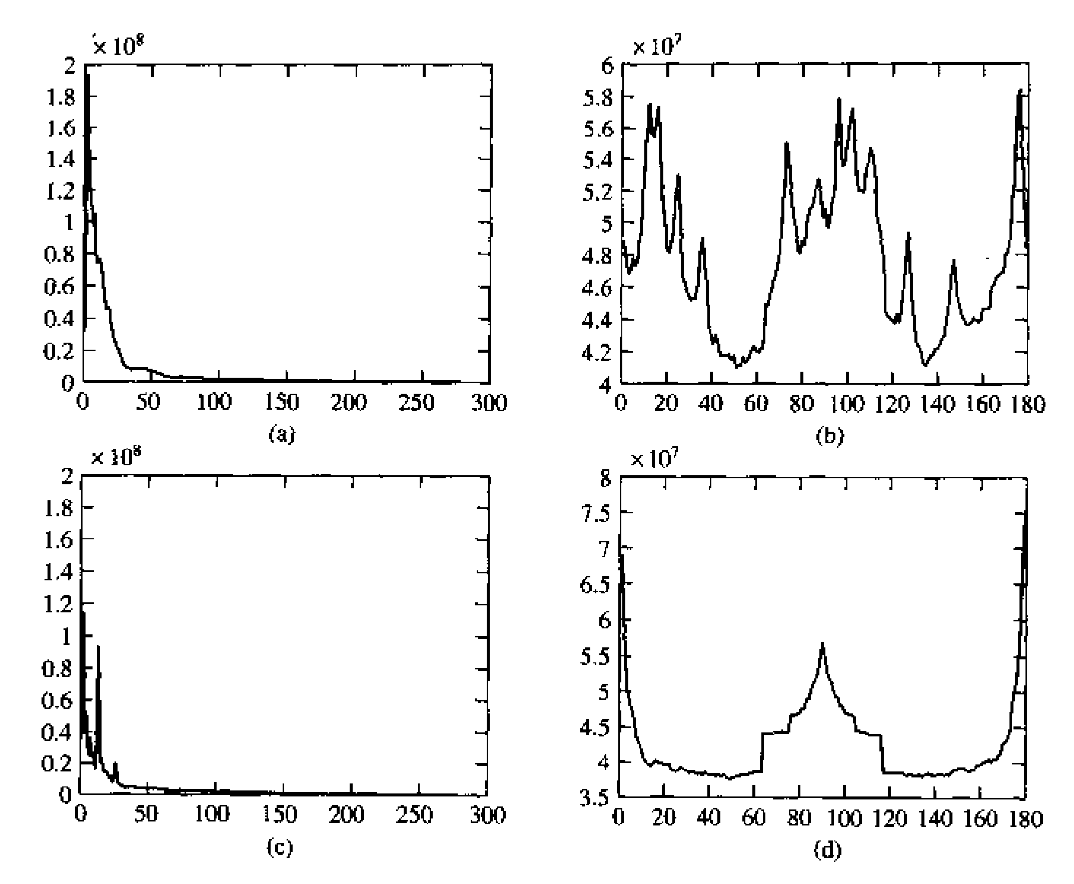


图 11.22　(a)随机排列图像的 $S(r)$曲线；(b)随机排列图像的 $S(\theta)$曲线；(c)有序排列图像的 $S(r)$曲线；(d)有序排列图像的 $S(\theta)$曲线

11.4.3　不变矩

一幅数字图像 $f(x, y)$的二维$(p+q)$阶矩定义为

$$m_{pq} = \sum_x \sum_y x^p y^q f(x, y)$$

其中，$p, q = 0, 1, 2, \cdots$，求和在跨越图像的所有空间坐标 x，y 的值上进行。相应的中心矩定义为

$$\mu_{pq} = \sum_x \sum_y (x - \overline{x})^p (y - \overline{y})^q f(x, y)$$

其中，

$$\overline{x} = \frac{m_{10}}{m_{00}} \quad , \quad \overline{y} = \frac{m_{01}}{m_{00}}$$

归一化$(p+q)$阶中心矩定义为

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}$$

其中 $p, q = 0, 1, 2, \cdots$，

$$\gamma = \frac{p+q}{2} + 1$$

其中 $p + q = 2, 3, \cdots$。

对平移、缩放、镜像和旋转都不敏感的 7 个二维不变矩的集合可以由这些公式推导出来，它们为

$$
\begin{aligned}
\phi_1 &= \eta_{20} + \eta_{02} \\
\phi_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
\phi_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
\phi_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
\phi_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 \\
&\quad -3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03}) \\
&\quad [3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
\phi_6 &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
&\quad + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
\phi_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 \\
&\quad - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{12} - \eta_{30})(\eta_{21} + \eta_{03}) \\
&\quad [3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]
\end{aligned}
$$

用于计算不变矩的M函数 invmoments 可直接实现这7个等式。该函数的语法为（其代码清单包含在附录C中）

```
phi = invmoments(f)
```

其中，f 为输入图像，phi 是一个包含有刚定义的不变矩的7元素行向量。

例 11.12　不变矩

图 11.23(a)所示的图像是使用如下命令从一幅大小为 400 × 400 像素的原图像获得的：

```
>> fp = padarray(f, [84 84], 'both');
```

使用零填充是为了使所有显示的图像与占有最大区域（568 × 568 个像素）的图像一致，如下面所讨论的那样，最大区域的图像已旋转了45°。填充仅用于显示目的，而不用于任何计算中。使用如下命令可获得半大图像以及相应的填充图像：

```
>> fhs = f(1:2:end, 1:2:end);
>> fhsp = padarray(fhs, [184 184], 'both');
```

使用 MATLAB 函数 fliplr 可获得镜像图像：

```
>> fm = fliplr(f);
>> fmp = padarray(fm, [84 84], 'both');
```

为旋转图像，我们可以使用函数 imrotate：

```
g = imrotate(f, angle, method, 'crop')
```

该函数逆时针旋转图像 angle 度。参数 method 可以是如下三者之一：

'nearest'　使用最邻近插值方法
'bilinear'　使用线性插值方法（典型选择）
'bicubic'　使用双三次插值方法

为了适应图像的旋转，填充方法自动增大了图像的尺寸。若参数中包含 'crop'，则旋转后的图像的中心部分会被修剪为与原图像相同的尺寸。默认情况下仅指定 angle，此时使用 'nearest' 插值，且不会发生图像修剪。

如下命令可以生成例子中的旋转图像：

```
>> fr2 = imrotate(f, 2, 'bilinear');
>> fr2p = padarray(fr2, [76 76], 'both');
>> fr45 = imrotate(f, 45, 'bilinear');
```

注意，因为最后这幅图像是这些图像中最大的图像，所以不需要填充。在两幅旋转过的图像中，0是由IPT在旋转处理中产生的。

刚才讨论的5幅图像的7个不变矩由如下命令产生：

```
>> phiorig = abs(log(invmoments(f)));
>> phihalf = abs(log(invmoments(fhs)));
>> phimirror = abs(log(invmoments(fm)));
>> phirot2 = abs(log(invmoments(fr2)));
>> phirot45 = abs(log(invmoments(fr45)));
```

注意，这里使用 `log` 的绝对值代替了不变矩值本身。使用 `log` 可以缩小动态范围，绝对值可以避免处理在计算负不变矩的 `log` 时产生的复数。因为我们感兴趣的通常是矩的不变性，而不是它们的符号，所以实践中通常使用绝对值。

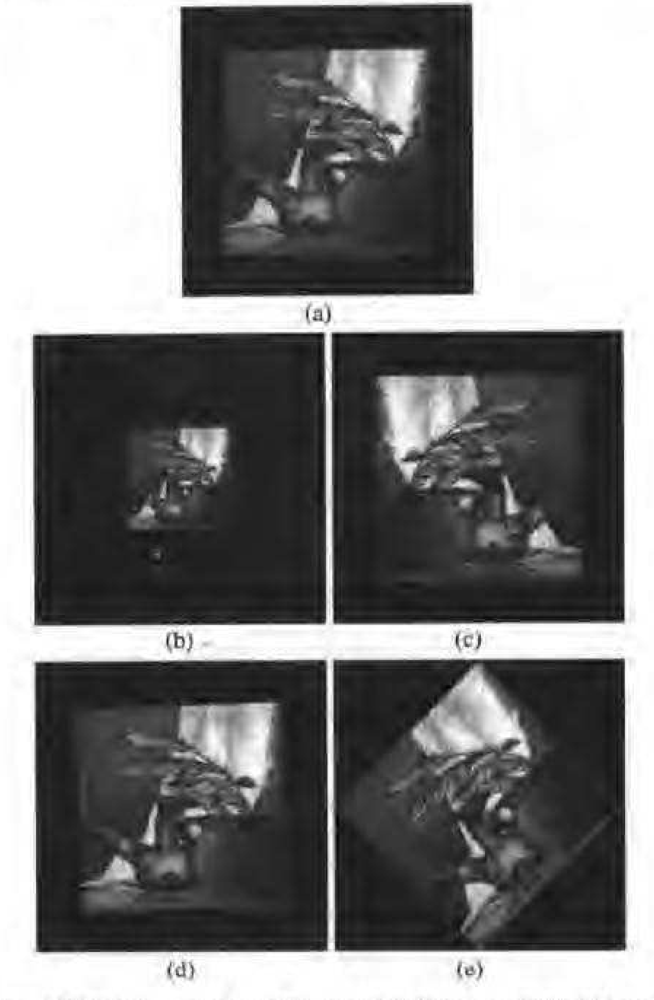


图 11.23 (a)经填充后的原图像；(b)半大图像；(c)镜像图像；(d)旋转2°后的图像；(e)旋转45°后的图像。在图(a)到图(d)中进行零填充的目的是产生与图(e)的大小相同的图像

原图像、半大图像、镜像图像和旋转图像的7个矩列在表11.4中。注意，越接近于这些数字，就越表明对刚提到的变化的高度不变性。类似的这些结果可以解释40多年来图像描述中的不变矩方法仍然是基本方法的原因。

表 11.4　图 11.23(a)到图 11.23(e)所示图像的 7 个不变矩。注意，在第一列使用了 log 的幅值

不变矩（|log|）	原图像	半大图像	镜像图像	旋转 2° 后的图像	旋转 45° 后的图像
ϕ_1	6.600	6.600	6.600	6.600	6.600
ϕ_2	16.410	16.408	16.410	16.410	16.410
ϕ_3	23.972	23.958	23.972	23.978	23.973
ϕ_4	23.888	23.882	23.888	23.888	23.888
ϕ_5	49.200	49.258	49.200	49.200	49.198
ϕ_6	32.102	32.094	32.102	32.102	32.102
ϕ_7	47.953	47.933	47.850	47.953	47.954

11.5　主分量描述

假设我们有 n 幅已配准的图像，它们的“堆叠”方式如图 11.24 所示。对于任意给定的坐标对 (i, j)，都有 n 个像素，每一幅图像在该位置上都有一个像素。这些像素以列向量的形式排列：

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

若这些图像的大小为 $M \times N$，则在 n 幅图像中，包含所有像素的 n 维向量总共有 MN 个。

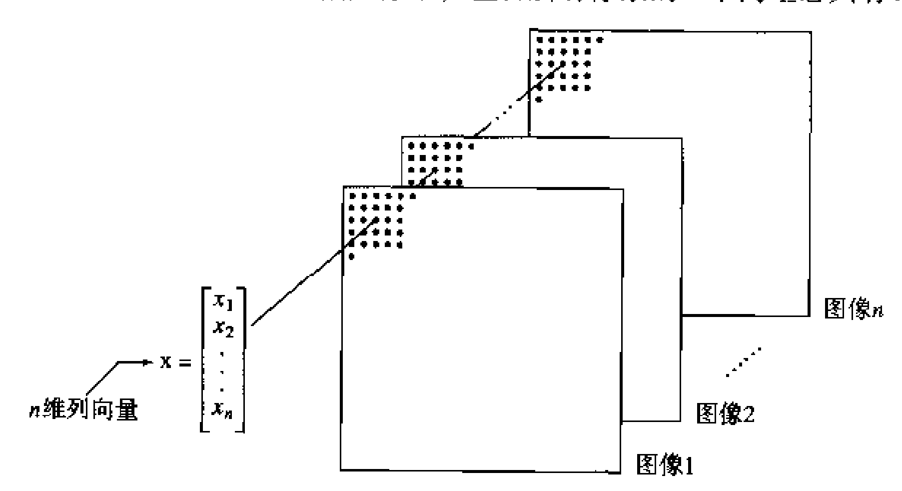


图 11.24　相同大小图像的堆叠中，由相应像素组成的向量

一个向量群的平均向量 $\mathbf{m}_\mathbf{x}$ 可以通过样本的平均值来近似：

$$\mathbf{m}_\mathbf{x} = \frac{1}{K}\sum_{k=1}^{K}\mathbf{x}_k$$

上式中，$K = MN$。类似地，向量群的 $n \times n$ 协方差矩阵 $\mathbf{C}_\mathbf{x}$ 可由下式近似计算：

$$\mathbf{C}_\mathbf{x} = \frac{1}{K-1}\sum_{k=1}^{K}(\mathbf{x}_k - \mathbf{m}_\mathbf{x})(\mathbf{x}_k - \mathbf{m}_\mathbf{x})^T$$

为了从样本值获得 $\mathbf{C}_\mathbf{x}$ 的无偏估计，这里用 $K-1$ 代替 K。因为 $\mathbf{C}_\mathbf{x}$ 是一个实对称矩阵，所以总可以找到 n 个正交特征向量。

主分量变换（也称为 Hotelling 变换）由下式给出：

$$\mathbf{y} = \mathbf{A}(\mathbf{x} - \mathbf{m}_\mathbf{x})$$

不难看出，向量y的元素是不相关的。因此，协方差矩阵$\mathbf{C}_y$是对角阵。矩阵A的行是$\mathbf{C}_x$的归一化特征向量。因为$\mathbf{C}_x$是一个实对称矩阵，这些向量构成了一个正交集，所以这些沿着$\mathbf{C}_y$主对角线方向的元素就是$\mathbf{C}_x$的特征值。$\mathbf{C}_y$的第i行主对角线元素是向量元素y_i的变量。

因为矩阵A的行向量是正交的，所以它的逆矩阵等于其转置。因此，可以通过求A的逆变换来获得x：

$$\mathbf{x} = \mathbf{A}^T\mathbf{y} + \mathbf{m}_x$$

当仅使用q个特征向量时，矩阵A是一个大小为$q \times n$的矩阵$\mathbf{A}_q$，这时主分量变换的重要性就显而易见了。现在，重构是一个近似值：

$$\hat{\mathbf{x}} = \mathbf{A}_q^T\mathbf{y} + \mathbf{m}_x$$

x的精确值和近似重构值之间的均方差为

$$\begin{aligned} e_{\text{ms}} &= \sum_{j=1}^{n}\lambda_j - \sum_{j=1}^{q}\lambda_j \\ &= \sum_{j=q+1}^{n}\lambda_j \end{aligned}$$

该等式的第一行表示，若$q = n$（即逆变换中使用了所有的特征向量），则误差为零。该等式也表明对$\mathbf{A}_q$选取最大特征值所对应的q个特征向量可以使误差最小。这样，在向量x及其近似值$\hat{\mathbf{x}}$之间的最小均方误差情况下，主分量变换是最优的。主分量变换得名于它使用了协方差矩阵的最大特征值所对应的特征向量。为进一步阐明这一概念，下面给出一个例子。

使用如下命令，一组n幅已配准的图像（每幅图像的大小均为$M \times N$）可转换为图11.24所示的堆叠形式：

```
>> S = cat(3, f1, f2,..., fn);
```

使用函数imstack2vectors（其代码请参阅附录C），我们可将这个大小为$M \times N \times n$的图像堆叠数组转换为一个数组，该数组的行是n维向量。该函数的语法为

```
[X, R] = imstack2vectors(S, MASK)
```

其中，S是一个图像堆栈，X是使用图11.24所示的方法从S中提取出来的向量数组。输入MASK是一幅大小为$M \times N$的逻辑图像或数字图像，该图像在用S的元素来生成X的位置有非零元素，而在将被忽略的位置元素为零。例如，若我们仅想在堆叠的这些图像的右上方象限使用向量，则在该象限中MASK的值为1，而在其余位置的值为0。若参数中未包含MASK，则所有的图像位置将用于形成X。最后，参数R是一个数组，该数组的行对应于用于形成X的向量的位置的二维坐标。在例12.2中，我们将显示如何使用MASK。在当前的讨论中，我们将使用默认值。

下面的M函数covmatrix用于计算平均向量和X中的向量的协方差矩阵。

```
function [C, m] = covmatrix(X)
%COVMATRIX Computes the covariance matrix of a vector population.
%   [C, M] = COVMATRIX(X) computes the covariance matrix C and the
%   mean vector M of a vector population organized as the rows of
%   matrix X. C is of size N-by-N and M is of size N-by-1, where N is
%   the dimension of the vectors (the number of columns of X).
[K, n] = size(X);
X = double(X);
```

```
if n == 1 % Handle special case.
   C = 0;
   m = X;
else
   % Compute an unbiased estimate of m.
   m = sum(X, 1)/K;
   % Subtract the mean from each row of X.
   X = X - m(ones(K, 1), :);
   % Compute an unbiased estimate of C. Note that the product is
   % X'*X because the vectors are rows of X.
   C = (X'*X)/(K - 1);
   m = m'; % Convert to a column vector.
end
```

下面的函数可实现本节所讨论的概念。注意，使用结构的目的在于简化输出变量。

```
function P = princomp(X, q)
%PRINCOMP Obtain principal-component vectors and related quantities.
%   P = PRINCOMP(X, Q) Computes the principal-component vectors of
%   the vector population contained in the rows of X, a matrix of
%   size K-by-n where K is the number of vectors and n is their
%   dimensionality. Q, with values in the range [0, n], is the number
%   of eigenvectors used in constructing the principal-components
%   transformation matrix. P is a structure with the following
%   fields:
%
%      P.Y        K-by-Q matrix whose columns are the principal-
%                 component vectors.
%      P.A        Q-by-n principal components transformation matrix
%                 whose rows are the Q eigenvectors of Cx corresponding
%                 to the Q largest eigenvalues.
%      P.X        K-by-n matrix whose rows are the vectors reconstructed
%                 from the principal-component vectors. P.X and P.Y are
%                 identical if Q = n.
%     P.ems       The mean square error incurred in using only the Q
%                 eigenvectors corresponding to the largest
%                 eigenvalues. P.ems is 0 if Q = n.
%     P.Cx        The n-by-n covariance matrix of the population in X.
%     P.mx        The n-by-1 mean vector of the population in X.
%     P.Cy        The Q-by-Q covariance matrix of the population in
%                 Y. The main diagonal contains the eigenvalues (in
%                 descending order) corresponding to the Q eigenvectors.

[K, n] = size(X);
X = double(X);

% Obtain the mean vector and covariance matrix of the vectors in X.
[P.Cx, P.mx] = covmatrix(X);
P.mx = P.mx'; % Convert mean vector to a row vector.

% Obtain the eigenvectors and corresponding eigenvalues of Cx. The
% eigenvectors are the columns of n-by-n matrix V. D is an n-by-n
% diagonal matrix whose elements along the main diagonal are the
```

其余5幅图像也可使用相同的方式来获得和显示。由于特征值在P.Cy的主对角线上，所以我们使用

```
>> d = diag(P.Cy);
```

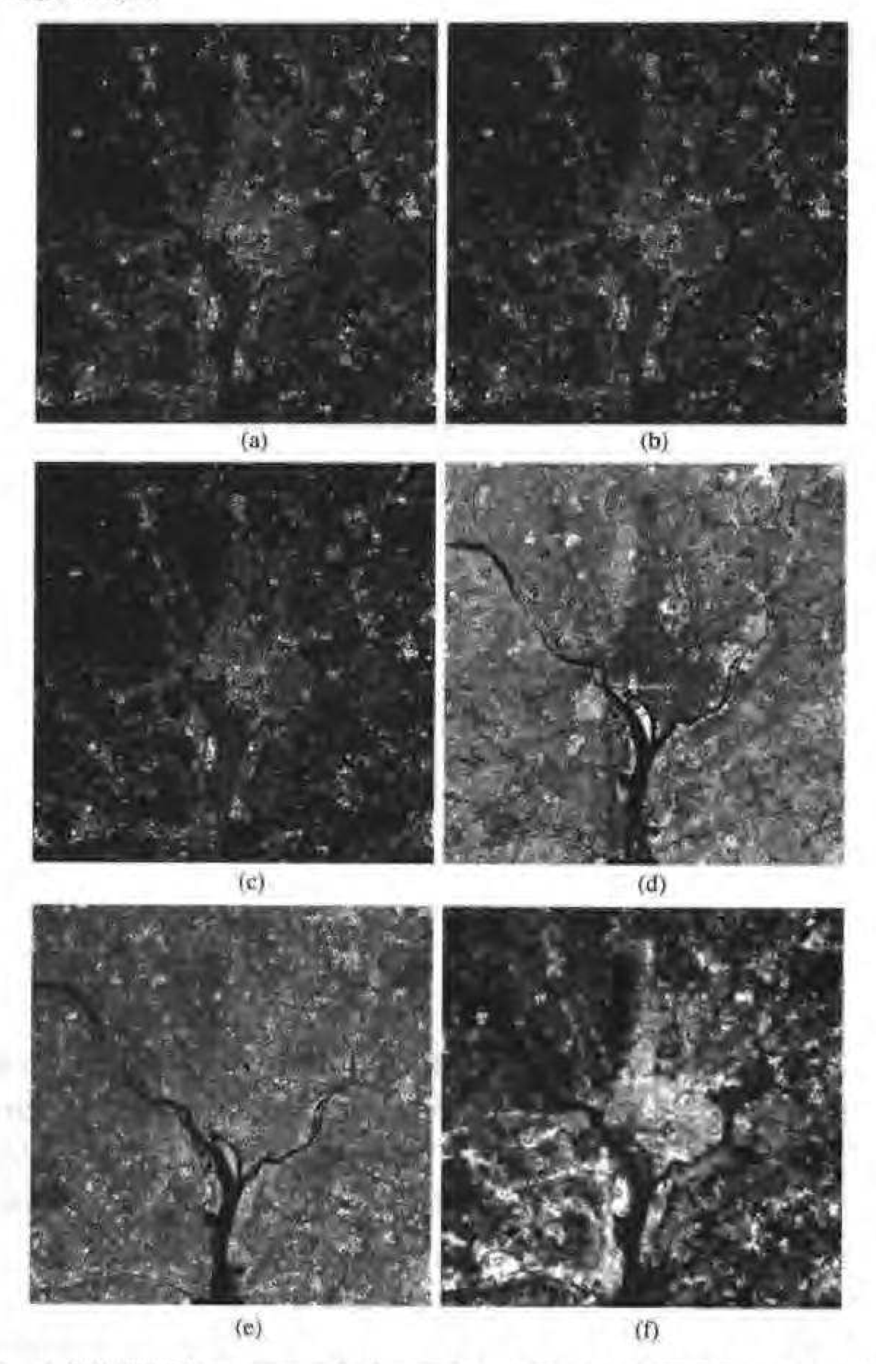


图11.25 六幅多谱图像：(a)可见蓝色波段图像；(b)可见绿色波段图像；(c)可见红色波段图像；(d)近红外光图像；(e)中红外光图像；(f)热红外波段图像（图像由NASA提供）

其中d是一个6维列向量，因为我们在函数中使用了q = 6。

图11.26显示了6幅刚刚计算的主分量图像。一个最明显的特征就是对比度细节的主要部分包含于前两幅图像中，而这两幅图像之后的几幅图像的对比度迅速下降。观察一下特征值就很容易解释其原因。如表11.5中的数值所示，前两个特征值与其他值相比要大很多。因为特征值

是向量**y**的元素的方差，而该方差是对比度的度量，因而与主特征值对应的图像显示很高的对比度是必然的。

假设我们使用小一些的 q 值，如 q = 2。则重构仅基于两幅主分量图像。为每幅图像使用

```
>> P = princomp(X, 2);
```

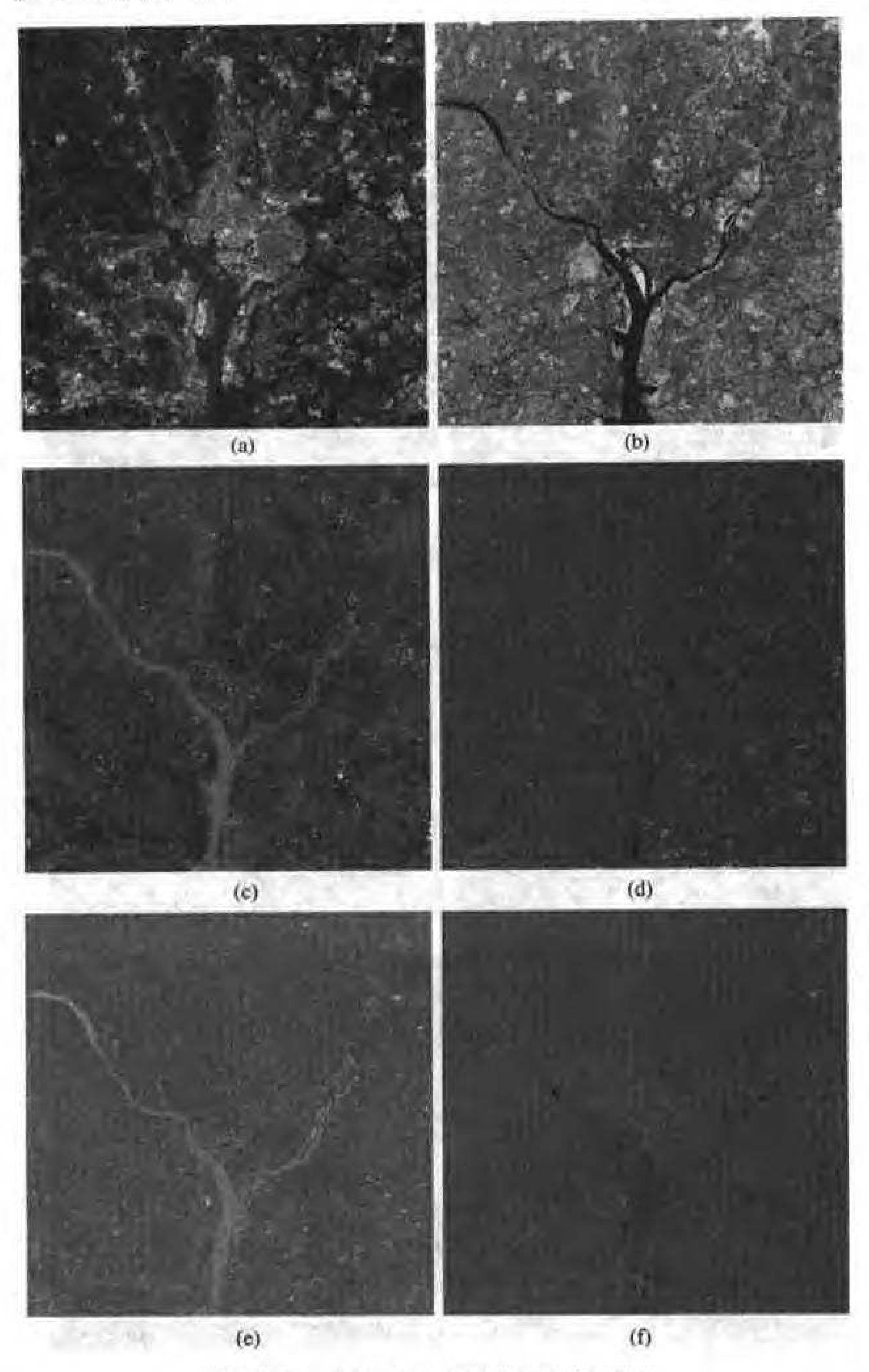
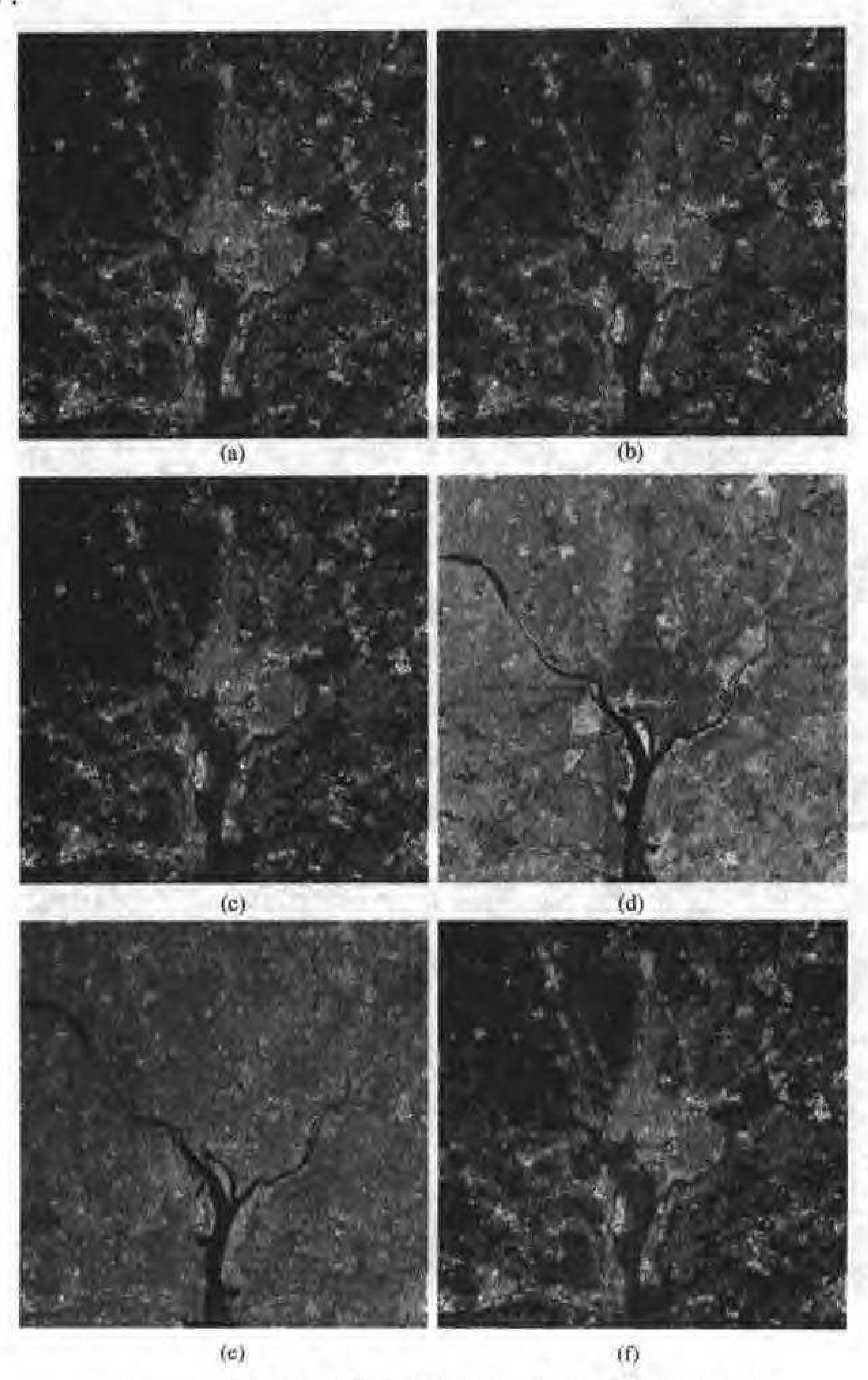


图 11.26 与图 11.25 相对应的主分量图像

和语句

```
>>h1 = P.X(:, 1);
>>h1 = reshape(h1, 512, 512);
```

可产生图 11.27 所示的重权图像。显然，这些图像相当接近于图 11.25 中所示的原图像。实际上，每幅图像均有不同程度的退化。例如，为比较原图像和重构的波段 1 的图像，我们可使用如下语句：



(a) (b) (c) (d) (e) (f)

图 11.27 仅使用有着最大主差的两幅主分量图像重构的多谱图像。请将它们与图 11.25 中的原图像进行比较

```
>>D1 = double(f1) - double(h1);
>>D1 = gscale(D1);
>>imshow(D1)
```

图11.28(a)显示了结果。这幅图像中的低对比度表明，当仅使用两幅主分量图像重构原图像时，丢失了不多的可视数据。图11.28(b)显示了波段6的图像的差异。这种差异是很明显的，因为原始的波段6的图像是很模糊的。但在重构图像中用到的两幅主分量图像则很清晰，且它们对重构有着很大的影响。仅使用两幅主分量图像时导致的均方差可由如下语句给出：

```
P.ems

ans =

    1.7311e+003
```

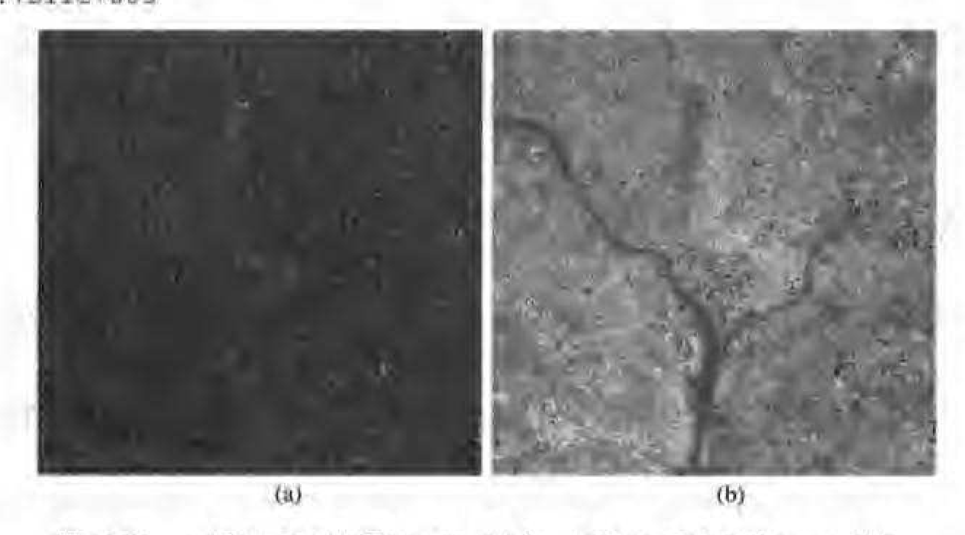


图11.28 (a)图11.27(a)与图11.25(a)的差；(b)图11.27(f)与图11.25(f)的差。两幅图像的8比特灰度级都缩放到了范围[0, 255]内

它是表11.5中的4个较小特征值的和。

表11.5 q = 6时 P.Cy 的特征值

λ_1	λ_2	λ_3	λ_4	λ_5	λ_6
10 352	2959	1403	203	94	31

在结束本节的讨论之前，我们要指出函数 princomp 可用于把对象（区域或边界）与对象的特征向量对齐。这些对象的坐标排列为 X 的列，且我们使用 q = 2。与特征向量对齐的变换后的数据包含在 P.Y 中。这是一个粗糙的对齐过程，该过程使用全部坐标来计算变换矩阵，并在其主扩展方向上对齐数据。

小结

从一幅图像中分割出的对象或区域的表示是准备图像数据的最早步骤。例如，描绘子就是下章中将开发的对象识别算法的输入。本章前几节中开发的M函数是对（用于图像表示与描述的）标准的IPT函数的重要扩展。描绘子类型的选择在很大程度上取决于所遇到的问题。这就是求解图像处理问题时要具有灵活的原型环境的原因，在该环境中，已有的函数可以集成新的代码，以便增加灵活性并减少开发时间。本章的内容就是如何构建这种环境的基础。

第12章　对象识别

前言

我们以讨论和开发几个关于区域或边界识别的M函数来结束本书。在本章中，这些区域或边界称为对象或模式。用计算机处理模式识别的方法可分为两个主要类别：决策理论方法和结构方法。第一类方法处理用定量描绘子描绘的模式，如长度、面积、纹理和在第11章讨论过的描绘子。第二类方法处理用符号信息表示的模式，如字符串和将在12.4节中介绍的字符属性及这些字符之间的关系。识别理论的核心就是从样本中"学习"的概念。决策理论方法和结构方法的学习技术将在下面几节中加以实现和说明。

12.1　背景知识

模式是第11章讨论过的那些描绘子的排列组合。在模式识别的文献中，特征常用于表示一个描绘子。模式分类就是一族具有相同属性的模式。模式分类用 $\omega_1, \omega_2, \cdots, \omega_W$ 来表示，其中 W 是分类数。用计算机进行模式识别涉及自动地或尽可能少地用手工方式将各种模式分配到它们各自的分类中的技术。

在实际应用中，两种主要的模式排列方法是向量法（用于定量描述）和字符串法（用于结构描述）。模式向量由粗体小写字母 **x**, **y**, **z** 表示，并具有 $n \times 1$ 维向量形式

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

其中，每个分量 x_i 表示第 i 个描绘子，n 是与模式有关的描绘子的总数。有时在计算中会用到 $1 \times n$ 维行向量，它可简单地由前面的列向量的转置 $\mathbf{x}^T$ 得到。

一个模式向量的**x**分量的性质依赖于描述物理模式本身的方法。例如，考虑字母符号的自动分类问题。适合于决策理论方法的描绘子可能包括用于描述字母外边界的二维不变矩或一组傅里叶变换系数的度量。

在某些应用中，模式特性用结构关系就可以很好地描述。例如，指纹识别是以称之为细节的纹路特征之间的关系为基础的。综合特征的相对尺寸和位置关系，这些特征是描述指纹特征（如断点、分支、合并与不连续部分）的基本要素。这类识别问题不仅要定量度量每个特征，而且还要度量这些特征的空间关系，以决定类成员，通常这类问题的最好解决方法是结构方法。

后面几节的内容是在MATLAB中解决模式识别问题的典型技术。在识别中，特别是在决策理论应用中，一个基本概念就是基于向量间的距离度量的模式匹配思想。因此，我们以在MATLAB中讨论各种有效计算距离度量的方法开始。

逆矩阵操作是计算Mahalanobis距离的最耗时计算任务。通过使用表2.4中介绍的MATLAB的矩阵右除操作符（/），我们可以有效地优化矩阵求逆操作。11.5 节中给出了 $\mathbf{m}_x$ 和 $\mathbf{C}_x$ 的表达式。

令 $\mathbf{X}$ 表示一族 p 个 n 维向量，$\mathbf{Y}$ 表示一族 q 个 n 维向量，在 $\mathbf{X}$ 和 $\mathbf{Y}$ 中的向量是这些数组的行。如下 M 函数的目标是计算 $\mathbf{Y}$ 和均值 $\mathbf{m}_x$ 之间的 Mahalanobis 距离。

```
function d = mahalanobis(varargin)
%MAHALANOBIS Computes the Mahalanobis distance.
%   D = MAHALANOBIS(Y, X) computes the Mahalanobis distance between
%   each vector in Y to the mean (centroid) of the vectors in X, and
%   outputs the result in vector D, whose length is size(Y, 1). The
%   vectors in X and Y are assumed to be organized as rows. The
%   input data can be real of complex. The outputs are real
%   quantities.
%
%   D = MAHALANOBIS(Y, CX, MX) computes the Mahalanobis distance
%   between each vector in Y and the given mean vector, MX. The
%   results are output in vector D, whose length is size(Y, 1). The
%   vectors in Y are assumed to be organized as the rows of this
%   array. The input data can be real or complex. The outputs are
%   real quantities. In addition to the mean vector MX, the
%   covariance matrix CX of a population of vectors X also must be
%   provided. Use function COVMATRIX (Section 11.5) to compute MX and
%   CX.

% Reference: Acklam, P. J. [2002]. "MATLAB Array Manipulation Tips
% and Tricks." Available at
%     home.online.no/~pjacklam/matlab/doc/mtt/index.html
% or at
%     www.prenhall.com/gonzalezwoodseddins

param = varargin; % Keep in mind that param is a cell array.
Y = param{1};
ny = size(Y, 1); % Number of vectors in Y.

if length(param) == 2
   X = param{2};
   % Compute the mean vector and covariance matrix of the vectors
   % in X.
   [Cx, mx] = covmatrix(X);
elseif length(param) == 3 % Cov. matrix and mean vector provided.
   Cx = param{2};
   mx = param{3};
else
   error('Wrong number of inputs.')
end
mx = mx(:)'; % Make sure that mx is a row vector.

% Subtract the mean vector from each vector in Y.
Yc = Y - mx(ones(ny, 1), :);

% Compute the Mahalanobis distances.
d = real(sum(Yc/Cx.*conj(Yc), 2));
```

在最后一行代码中调用real的目的是消除“数字噪声”，就像我们在第4章中对一幅图像滤波后所做的那样。若我们知道数据总是实数，则可以通过删除函数 real 和 conj 来简化代码。

12.3 基于决策理论方法的识别

识别的决策理论方法基于决策（也称为判别）函数的使用。如在12.1节中讨论的那样，令 $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$ 表示一个 n 维模式向量。对于 $\omega_1, \omega_2, \cdots, \omega_W$ 个模式分类，决策理论模式识别的基本问题是寻找 W 个决策函数 $d_1(\mathbf{x}), d_2(\mathbf{x}), \cdots, d_W(\mathbf{x})$，这些决策函数要具有如下性质：若模式 $\mathbf{x}$ 属于 ω_i 类，则

$$d_i(\mathbf{x}) > d_j(\mathbf{x}) \qquad j = 1, 2, \cdots, W; j \neq i$$

换言之，若将 $\mathbf{x}$ 代入所有的决策函数 $d_i(\mathbf{x})$ 中后可产生最大的数值，则称该未知的模式 $\mathbf{x}$ 属于第 i 个模式分类。

分隔类 ω_i 与类 ω_j 的决策边界由满足关系 $d_i(\mathbf{x}) = d_j(\mathbf{x})$ 的 $\mathbf{x}$ 值给出，或由满足下式的 $\mathbf{x}$ 值给出：

$$d_i(\mathbf{x}) - d_j(\mathbf{x}) = 0$$

实际情况下，我们通常用单个函数 $d_{ij}(\mathbf{x}) = d_i(\mathbf{x}) - d_j(\mathbf{x}) = 0$ 来表示两个分类间的决策边界。因此，对于类 ω_i 的模式，有 $d_{ij}(\mathbf{x}) > 0$，而对于类 ω_j 有 $d_{ij}(\mathbf{x}) < 0$。

在接下来的几节中，我们将会了解到决策函数的寻找需要从我们感兴趣的分类的典型模式中进行参数估计。用于参数估计的模式称为“训练模式”或“训练集”。不用于训练而用来测试某一识别方法性能的已知模式集合，称为测试模式或独立模式，或独立集。12.3.2节和12.3.4节的主要目的是开发各种方法，以便通过从训练集使用参数估计来寻找决策函数。12.3.3节将使用相关方法来处理匹配问题，相关方法虽然可以表示成决策函数的形式，但传统上还是表示为直接图像匹配的形式。

12.3.1 形成模式向量

正如在本章开头指出的那样，模式向量可以由定量的描绘子形成，如在第11章中为区域和/或边界讨论的那些描绘子。例如，若我们使用傅里叶描绘子来描述一个边界，则第 i 个描绘子的值将作为模式向量的第 i 个分量 x_i 的值。我们还可以将其他分量添加到模式向量中去，如在表11.2中，通过将纹理的6个度量值添加到每个向量中，我们可将6个附加分量合并到傅里叶描绘子中。

在处理（已配准的）多谱图像时，另一种频繁使用的方法是堆叠这些图像，然后由图像中的相应像素形成向量，如图11.24所示。这些图像是使用函数 cat 函数来堆叠的：

```
S = cat(3, f1, f2, ..., fn)
```

其中，S 是堆叠，f1，f2，...，fn 是形成堆叠的图像。然后使用11.5节中讨论过的函数 imstack2vectors 来生成向量。详情请参见例12.2。

12.3.2 使用最小距离分类器的模式匹配

假设每个模式分类 ω_j 均由均值向量 $\mathbf{m}_j$ 来表征，即用每个训练向量族的均值向量来表示该类向量：

$$\mathbf{m}_j = \frac{1}{N_j} \sum_{\mathbf{x} \in \omega_j} \mathbf{x} \qquad j = 1, 2, \cdots, W$$

其中，N_j是来自分类ω_j的训练模式向量的个数，求和就在这些向量上进行。与前面一样，W是模式分类的个数。决定未知模式向量$\mathbf{x}$属于哪一类的一种方法是将其指定到距它最近的原型分类中。通过将欧几里得距离用做邻近性（即相似性）的度量，可将该问题约简为计算距离度量问题：

$$D_j(\mathbf{x}) = \|\mathbf{x} - \mathbf{m}_j\| \qquad j = 1, 2, \cdots, W$$

若$D_i(\mathbf{X})$是最小距离，则我们将$\mathbf{x}$指派为类ω_i。换言之，最小距离在该式中意味着最佳匹配。

假设所有均值向量都组织为一个矩阵$\mathbf{M}$的行，则任一模式$\mathbf{x}$到所有均值向量的距离的计算可以使用12.2节中讨论的表达式实现：

```
d = sqrt(sum(abs(M - repmat(x, W, 1)).^2, 2))
```

因为所有的距离均为正值，所以这条语句可通过忽略`sqrt`操作来简化。`d`的最小值决定了模式向量`x`的分类成员：

```
>> class = find(d == min(d));
```

换言之，若`d`的最小值在它的第k个位置（即`x`属于第`k`个模式分类），则标量`class`将等于`k`。若存在一个以上的最小值，则`class`就等于一个向量，该向量中的每个元素指向不同最小值的位置。

若我们有一组排列为矩阵`X`的行的模式面不是单个模式，则我们可使用一个表达式（该表达式类似于12.2节中的较长表达式）来得到一个矩阵`D`，该矩阵的元素`D(I,J)`是`X`中的第i个模式向量与`M`中的第j个均值向量之间的欧几里得距离。这样，要寻找`X`中的第i个模式所属的分类，只要寻找在`D`的第i行中最小值所处的列的位置即可。若出现多个最小值的情况，则情形类似于上一段讨论的单个向量。

不难看出，选择最小距离等价于求下面的函数值：

$$d_j(\mathbf{x}) = \mathbf{x}^T\mathbf{m}_j - \frac{1}{2}\mathbf{m}_j^T\mathbf{m}_j \qquad j = 1, 2, \cdots, W$$

若$d_i(\mathbf{x})$出现了最大的数值，则将$\mathbf{x}$指派到类ω_i。该公式与早先定义的决策函数的概念是一致的。

对于最小距离分类器，类ω_i与类ω_j之间的决策边界为

$$\begin{aligned} d_{ij}(\mathbf{x}) &= d_i(\mathbf{x}) - d_j(\mathbf{x}) \\ &= \mathbf{x}^T(\mathbf{m}_i - \mathbf{m}_j) - \frac{1}{2}(\mathbf{m}_i - \mathbf{m}_j)^T(\mathbf{m}_i + \mathbf{m}_j) = 0 \end{aligned}$$

由该式给出的表面是连接$\mathbf{m}_i$与$\mathbf{m}_j$的线段的中垂线。当$n = 2$时，该中垂线为一条直线；当$n = 3$时，它是一个平面；当$n > 3$时，它是一个超平面。

12.3.3 相关匹配

相关通常很简单。给定一幅图像$f(x, y)$，相关问题就是在该图像中寻找与事先给定的子图像（也称为掩膜或模板）$w(x, y)$相匹配的所有位置。典型情况下，$w(x, y)$总是比$f(x, y)$小得多。寻找匹配的一种方法是，将$w(x, y)$作为一个空间滤波器来在f中的每个位置计算w与f的乘积的和（或归一化后的值），这与3.4.1节解释的方式完全一样。$w(x, y)$在$f(x, y)$中的最佳匹配是在得到的相关图像中出现最大值的位置。除非$w(x, y)$很小，否则刚刚描述的方法的计算量一般都会较大。因此，空间相关在实际实现中总是依靠硬件来实现的。

对于原型开发，一种变通的方法是在频率域实现相关，这与我们第4章讨论的卷积理论相似，用相关理论可以将空间相关与图像变换的乘积联系起来。若我们用“∘ ”表示相关，用“*”表示复共轭，则相关定理陈述为

$$f(x,y)\circ w(x,y) \Leftrightarrow F(u,v)H^*(u,v)$$

换言之，空间相关可以用一个函数的傅里叶变换与另一个函数的傅里叶变换的复共轭的乘积的傅里叶逆变换得到。该式反过来也成立：

$$f(x,y)w^*(x,y) \Leftrightarrow F(u,v)\circ H(u,v)$$

相关理论的第二个性质是其内在完备性，但在本章中不会用到它。

以M函数的形式来实现第一个相关结果是非常简单的，代码如下所示：

```
function g = dftcorr(f, w)
%DFTCORR 2-D correlation in the frequency domain.
%   G = DFTCORR(F, W) performs the correlation of a mask, W, with
%   image F. The output, G, is the correlation image, of class
%   double. The output is of the same size as F. When, as is
%   generally true in practice, the mask image is much smaller than
%   G, wraparound error is negligible if W is padded to size(F).

[M, N] = size(f);
f = fft2(f);
w = conj(fft2(w, M, N));
g = real(ifft2(w.*f));
```

例12.1 使用相关来匹配图像

图12.1(a)显示了一幅"安德鲁"飓风的图像，其中的风眼清晰可见。作为相关的一个实例，我们希望在图12.1(a)中找到与图12.1(b)的暴风眼图像最佳匹配的位置。图像大小为912 × 912像素，掩模大小为32 × 32像素。图12.1(c)是使用下列命令后的结果：

```
>> g = dftcorr(f, w);
>> gs = gscale(g);
>> imshow(gs)
```

图12.1(c)所示的相关图像比较模糊，但这并不奇怪，因为图12.1(b)中的图像主要由两个几乎不变的区域组成，因而它的作用就等于一个低通滤波器。

我们感兴趣的特征是最佳匹配的位置，对于相关运算，这意味着要从相关图像中寻找数值最大的位置：

```
>> [I, J] = find(g == max(g(:)))
I =
    554
J =
    203
```

在这个例子中，最大值是惟一的。正如3.4.1节中所述的那样，相关图像的坐标与模板的移动相对应，因此，坐标[I,J]对应于模板左下角的位置。若模板放置在图像的上部，则我们将会发现在此坐标处模板与风眼十分接近。另一种寻找匹配位置的办法是在相关图像最大值的附近设定一个阈值，或者设定经比例缩放后的阈值gs，已知gs的最大值为255。例如，图12.1(d)中的图像是由如下命令得到的：

```
>> imshow(gs > 254)
```

在图12.1(d)中，带有小白点的模板的左下角再次说明最佳匹配是在风眼附近。

其中的对数可保证为实数，因为$p(\mathbf{x}/\omega_j)$和$P(\omega_j)$均非负。将高斯PDF代入可得如下等式：

$$d_j(\mathbf{x}) = \ln P(\omega_j) - \frac{n}{2}\ln 2\pi - \frac{1}{2}\ln|\mathbf{C}_j| - \frac{1}{2}[(\mathbf{x} - \mathbf{m}_j)^T \mathbf{C}_j^{-1}(\mathbf{x} - \mathbf{m}_j)]$$

项$(n/2)\ln 2\pi$对所有类别均是一样的正常数，因此可以将它去掉，从而该决策函数变为

$$d_j(\mathbf{x}) = \ln P(\omega_j) - \frac{1}{2}\ln|\mathbf{C}_j| - \frac{1}{2}[(\mathbf{x} - \mathbf{m}_j)^T \mathbf{C}_j^{-1}(\mathbf{x} - \mathbf{m}_j)]$$

其中，$j = 1, 2, \cdots, W$。可以看出，方括号内的项是我们在12.2节中讨论过的Mahalanobis距离，对它我们有一个向量化实现。在11.5节中，我们还将介绍一种计算均值和协方差矩阵的有效方法，采用这种方法时，在多变量高斯情形下，实现贝叶斯分类器是相当简单的，如以下函数所示：

```
function d = bayesgauss(X, CA, MA, P)
%BAYESGAUSS Bayes classifier for Gaussian patterns.
%   D = BAYESGAUSS(X, CA, MA, P) computes the Bayes decision
%   functions of the patterns in the rows of array X using the
%   covariance matrices and and mean vectors provided in the arrays
%   CA and MA. CA is an array of size n-by-n-by-W, where n is the
%   dimensionality of the patterns and W is the number of
%   classes. Array MA is of dimension n-by-W (i.e., the columns of MA
%   are the individual mean vectors). The location of the covariance
%   matrices and the mean vectors in their respective arrays must
%   correspond. There must be a covariance matrix and a mean vector
%   for each pattern class, even if some of the covariance matrices
%   and/or mean vectors are equal. X is an array of size K-by-n,
%   where K is the total number of patterns to be classified (i.e.,
%   the pattern vectors are rows of X). P is a 1-by-W array,
%   containing the probabilities of occurrence of each class. If
%   P is not included in the argument list, the classes are assumed
%   to be equally likely.
%
%   The output, D, is a column vector of length K. Its Ith element is
%   the class number assigned to the Ith vector in X during Bayes
%   classification.

d = [ ]; % Initialize d.
error(nargchk(3, 4, nargin)) % Verify correct no. of inputs.
n = size(CA, 1);             % Dimension of patterns.

% Protect against the possibility that the class number is
% included as an (n+1)th element of the vectors.
X = double(X(:, 1:n));
W = size(CA, 3); % Number of pattern classes.
K = size(X, 1);  % Number of patterns to classify.
if nargin == 3
   P(1:W) = 1/W; % Classes assumed equally likely.
else
   if sum(P) ~= 1
      error('Elements of P must sum to 1.');
   end
end

% Compute the determinants.
for J = 1:W
   DM(J) = det(CA(:, :, J));
end
```

```
% Compute inverses, using right division (IM/CA), where IM =
% eye(size(CA, 1)) is the n-by-n identity matrix. Reuse CA to
% conserve memory.
IM = eye(size(CA,1));
for J = 1:W
   CA(:, :, J) = IM/CA(:, :, J);
end

% Evaluate the decision functions. The sum terms are the
% Mahalanobis distances discussed in Section 12.2.
MA = MA'; % Organize the mean vectors as rows.
for I = 1:K
   for J = 1:W
      m = MA(J, :);
      Y = X - m(ones(size(X, 1), 1), :);
      if P(J) == 0
         D(I, J) = -Inf;
      else
         D(I, J) = log(P(J)) - 0.5*log(DM(J)) ...
                   - 0.5*sum(Y(I, :)*(CA(:, :, J)*Y(I, :)'));
      end
   end
end

% Find the maximum in each row of D. These maxima
% give the class of each pattern:
for I = 1:K
   J = find(D(I, :) == max(D(I, :)));
   d(I, :) = J(:);
end

% When there are multiple maxima the decision is
% arbitrary. Pick the first one.
d = d(:, 1);
```

例 12.2　多谱数据的贝叶斯分类

贝叶斯识别经常用于自动地分类多谱图像中的区域。图 12.2 显示了图 11.25 的前 4 幅图像（三幅可见光波段和一幅红外波段）。作为简单的示例，我们对这些图像中的三种类型的区域（分类）应用贝叶斯分类方法：水、市区和植被。该例中的模式向量是使用 11.5 节和 12.3.1 节中讨论的方法形成的，在这种方法中，图像中相应的像素被组织成向量。因为我们正处理 4 幅图像，所以模式向量是四维向量。

为获得均值向量和协方差矩阵，我们需要代表每种模式分类的样本。交互地获得这些样本的一种简单方法是使用函数 roipoly（见 5.2.4 节），该函数的语法为

```
>> B = roipoly(f);
```

其中，f 为任意一幅多谱图像，B 是一幅二值掩膜图像。采用这种格式，图像 B 交互地在屏幕上生成。图 12.2(e)显了使用该方法生成的三幅掩膜图像 B1、B2 和 B3 的合成图像。数字 1、2 和 3 分别标示了对应于水、城市开发区和植被的样本区域。

下一步，我们得到与每个区域相对应的向量。因为这 4 幅图像已在空间上配准，所以我们可以简单地沿着第三个方向级联这些图像，以便获得一个图像堆叠：

```
>> stack = cat(3, f1, f2, f3, f4);
```

其中，f1 到 f4 是图 12.2(a)到图 12.2(d)所示的 4 幅图像。在观察这 4 幅图像时，其中的任意一点都与一个四维模式向量相对应（见图 11.24）。我们对含有图 12.2(e)所示的三个区域的向量感兴趣，这些向量是使用 11.5 节中讨论的函数 imstack2vectors 得到的：

```
>> [X, R] = imstack2vectors(stack, B);
```

其中，X 是一个其行为向量的数组，R 也是一个数组，该数组的行是对应于 X 中的那些向量的位置（二维区域坐标）。

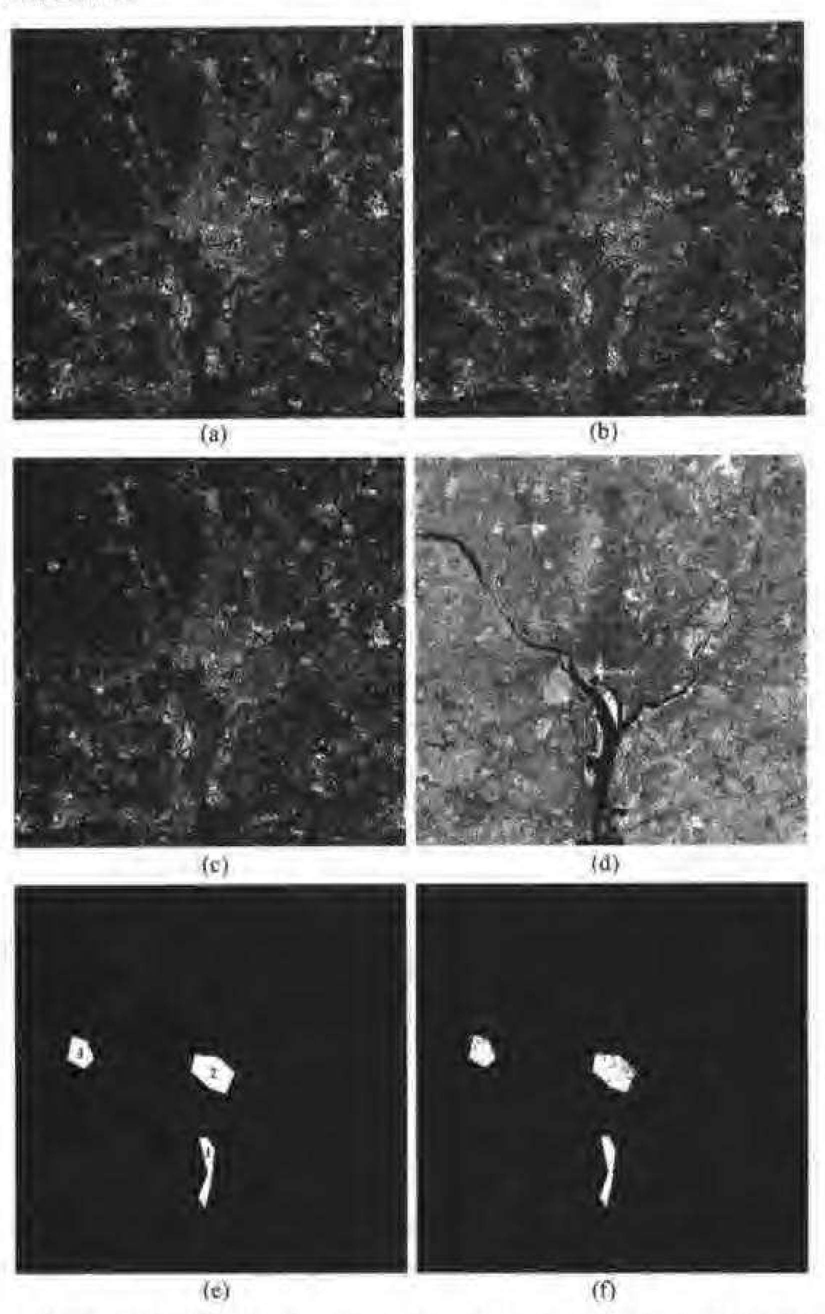


图 12.2　多谱数据的贝叶斯分类：(a)～(c)蓝、绿、红可见波长图像；(d)红外图像；(e)显示水的样本区域的掩模(1)，显示城市开发区的样本区域的掩模(2)以及显示植被的样本区域的掩模(3)；(f)分类结果。黑点表示分类不正确的点，区域中的其他（白色）点是分类正确的点（原图像由 NASA 提供）

使用imstack2vectors和三个掩膜B1、B2和B3，产生了三个向量集X1、X2和X3以及三个坐标集R1、R2和R3。然后从X中提取三个子集Y1、Y2和Y3，用于估计协方差矩阵和均值向量的训练样本。这些子集Y是通过对X1、X2和X3每隔一行取样产生的。Y1中向量的协方差矩阵和均值向量是用如下命令获得的：

```
>> [C1, m1] = covmatrix(Y1);
```

其他两个分类的情形类似。然后，我们使用如下命令来得到bayesgauss中所用的数组CA和MA：

```
>> CA = cat(3, C1, C2, C3);
>> MA = cat(2, m1, m2, m3);
```

带训练模式的分类器的性能通过分类训练集来确定：

```
>> dY1 = bayesgauss(Y1, CA, MA);
```

其他两个分类与此类似。分类1的错误分类模式的数目由如下语句得到：

```
>> IY1 = find(dY1 ~= 1);
```

找到其模式被错误分类的类很简单。例如，length(find(dY1 == 2)给出了从分类1错分类到分类2的模式数。对于其他模式集，我们可用相同的方法来处理。

表12.1总结了用训练模式集和独立模式集获得的识别结果。训练模式集和独立模式集正确识别的百分比大致相同，这也说明了参数估计的稳定性。在这两种情况下，最大误差源自市区所用的模式，这并不奇怪，因为市区中也存在植被（注意，没有市区或植被区域模式被错分为水）。图12.2(f)中，每个区域内以黑点表示的点代表错误分类，而白点则代表正确分类。在区域1中没有黑点，因为那7个错分点要么非常靠近，要么就在白色区域的边界上。

当然，要设计一个可操作的多谱分类的识别系统，我们还需要做其他的工作，但本例的重要性在于，用MATLAB和IPT函数，再加上本书前面开发的一些函数，可以轻松地对该系统建模。

表12.1 多谱图像数据的贝叶斯分类

训练模式						独立模式					
分类	样本数	分类类别			%	分类	样本数	分类类别			%
		1	2	3	正确率			1	2	3	正确率
1	484	482	2	0	99.6	1	483	478	3	2	98.9
2	933	0	885	48	94.9	2	932	0	880	52	94.4
3	483	0	19	464	96.1	3	482	0	16	466	96.7

12.3.5 自适应学习系统

12.3.1节和12.3.3节中讨论的方法是以用样本模式估计每个模式分类的统计参数为基础的。最小距离分类器由每个分类的均值向量指定。类似地，对高斯族的贝叶斯分类器，则完全由每一分类模式的均值向量和协方差矩阵指定。

在这两种方法中，训练很简单。每一类的训练模式被用来计算对应于该分类的决策函数的参数。一旦估计出参数，分类器的结构也就确定了，该分类器的最终性能将取决于实际模式集合与导出所用分类方法时做出的基本统计假设的符合程度。

只要模式分类具有或至少近似具有高斯概率密度函数的特性，刚才讨论的方法就非常有效。然而，若这个假设不成立，则设计一个统计分类器就变得很困难，因为估计多变量的概率密度函数不是一件简单的事情。在实际中，这些决策理论问题的最好解决方法是直接通过训练来产生需要的决策函数。这样一来，就没有必要假设基本模式分类的概率密度函数或其他的概率信息。

关于这一类的分类问题，现在所用的基本方法是基于神经网络的（Gonzalez and Woods[2002]）。适合于图像处理应用的神经网络适用范围其实并没有超过MATLAB和IPT中可用函数的能力。然而，在现有情况下，这一努力并无太大的效果，因为MathWorks综合神经网络工具箱已经使用许多年了。

12.4 结构识别

结构识别技术一般是将感兴趣的物体表示为串、树或图，然后再定义基于这些表示的描绘子和识别规则。决策理论方法与结构方法的关键区别在于，前者使用表示为数值向量形式的定量描绘子，而后者主要处理符号信息。例如，在给定的应用中，假设物体边界用最小周长的多边形表示。一种决策理论方法可能基于形成的向量，这些向量的元素是该多边形的内角值；而一种结构方法则可能基于定义的一些符号来排列角度值，然后形成这些符号的串来描述模式。

串是结构识别中用得最多的通用表示方法，因此本节主要讲述这一方法。我们在后面将会看到，MATLAB拥有一个专门用于串操作的扩展函数集。

12.4.1 MATLAB中的串操作

在MATLAB中，一个串是一个一维数组，其中的元素是串中字符的数值码。显示的字符依赖于给定字符编码的字符集。串的长度是串中字符的个数，包括空格。串的长度由函数`length`获得。一个串由一对单引号和内部的字符来定义（串内文本中用到的引号，我们用双引号表示）。

表12.2列出了串操作的主要MATLAB函数[1]。我们首先考虑一般类型，函数`blanks`的语法为

```
s = blanks(n)
```

它产生了一个由n个空格组成的串。函数`cellstr`由一个字符数组生成一个单元数组。在单元数组中，存储字符串的主要优点之一是不需要用空格添加字符串来创建具有相同长度的行的字符数组（如执行串比较）。语法

```
c = cellstr(S)
```

将字符数组S的行放入c的独立单元。函数`char`将单元数组转换成串矩阵。例如，考虑串矩阵

```
>> S = [' abc'; 'defg'; 'hi  ']   % Note the blanks.
S =
 abc
defg
hi
```

① 本节讨论的某些串函数已在前面的章节介绍过。

表 12.2 MATLAB的串处理函数

类别	函数名	解释
普通	blanks	空串
	cellstr	从字符数组建立串的单元数组。使用函数 char 转换成一个字符串
	char	创建字符数组（串）
	deblank	消除后面的空格
	eval	用 MATLAB 表达式执行串
串测试	iscellstr	串的单元数组为真
	ischar	字符数组为真
	isletter	字母表的字母为真
	isspace	空白字符为真
串操作	lower	把串变为小写
	regexp	匹配规则表达式
	regexpi	匹配规则表达式，忽略大小写
	regexprep	用规则表达式代替串
	strcat	级联串
	strcmp	比较串（见 2.10.5 节）
	strcmpi	比较串，忽略大小写
	strfind	在其他串中寻找一个串
	strjust	对齐串
	strmatch	寻找匹配串
	strncmp	比较串的前 n 个字符
	strncmpi	比较串的前 n 个字符，忽略大小写
	strread	从串中读取格式化的数据，详见 2.10.5 节
	strrep	用其他串代替一个串
	strtok	在串中寻找标记
	strvcat	纵向级联串
	upper	将串转换为大写
串数转换	double	将串转换为数值码
	int2str	将整数转换为串
	mat2str	将矩阵转换为适合用 eval 函数处理的串
	num2str	将数转换为串
	sprintf	将格式化数据写成串
	str2double	将串转换为双精度值
	str2num	将串转换为数（见 2.10.5 节）
	sscanf	在格式控制之下读串
数字转换	base2dec	将 B 进制串转换为十进制整数
	bin2dec	将二进制串转换为十进制整数
	dec2base	将十进制整数转换为 B 进制串
	dec2bin	将十进制整数转换为二进制串
	dec2hex	将十进制整数转换为十六进制串
	hex2dec	将十六进制串转换为十进制整数
	hex2num	将 IEEE 十六进制数转换为双精度数

在提示符后键入 whos S 后，会显示如下信息：

```
>> whos S
    Name        Size        Bytes       Class
    S           3x4         24          char array
```

（续表）

元字符	用途
chars$	当一个串用 chars 结束时匹配
\<chars	当一个字用 chars 开始时匹配
chars\>	当一个字用 chars 结束时匹配
\<word\>	精确字匹配

本次讨论中，“字”是一个字符串内的子串，串的前面或开始处加有空格，并且以空格结束或在字符串的尾部加空格。下一段中给出了几个例子。

函数 regexp 匹配一个规则表达式。基本语法

```
idx = regexp(str, expr)
```

将返回一个行向量 idx，它包含在 str 中与规则表达式串 expr 相匹配的子串的索引（位置）。例如，假设 expr = 'b.*a'，则表达式 idx = regexp(str,expr) 意味着在 str 串中对任何 b 寻找匹配，b 后跟有任何字符（由元字符“.”决定）任意次，包括 0 次（由 * 指定），最后跟一个字符 a。在 str 中满足这些条件的位置索引都存放在向量 idx 中。若没有找到这样的位置，则 idx 作为空矩阵返回。

还有一些关于 expr 规则表达式的例子也可以阐明这些概念。规则表达式 'b.+a' 除了用“一次或一次以上”代替“可以重复任意次，包括 0 次”以外，与前面的例子是一样的。表达式 'b[0-9] ' 意味着 b 后面跟随着从 0 到 9 的任意数字；表达式 'b[0-9] * ' 意味着 b 后面跟有从 0 到 9 的任意数任意次；'b[0-9] + ' 表示 b 后面跟有从 0 到 9 的任意数一次或多次。例如，若 str = 'b0123c234bcd'，则上述三个 expr 将给出下面的结果：idx = 1；idx = [1 10] 和 idx = 1。

作为一个使用规则表达式识别对象特征的例子，假设一个对象的边界已经用 4 方向 Freeman 链码进行了编码［见图 11.1(a)］，并存放到串 str 中，则

```
>> str
str =
000300333222221111
```

再假设我们感兴趣的是在串中寻找位置，行进的方向从东（0）转到南（3），且保持在那里至少两个增量但不多于 6 个增量。这是对象中一个“向下一步”的特征，该特征由于噪声的缘故可能要比单次平移大。我们可以使用如下规则表达式来表示这些要求：

```
>> expr = '0[3]{2, 6}';
```

从而

```
>> idx = regexp(str, expr)
idx =
     6
```

在本例中，idx 的值指出了一个 0 后跟有三个 3 的点。用相似的方式可以构成更为复杂的表达式。

除了忽略字符大写或小写外，函数 regexpi 的工作原理与 regexp 相同。函数 regexprep 的语法为

```
C = regexprep(str, expr, replace)
```

该函数可以将 str 串中所有的规则表达式 expr 用串 replace 来替换。返回一个新串。若未发现匹配，则 regexprep 返回没有变化的字符串 str。

函数 strcat 的语法为

```
C = strcat(S1, S2, S3, ...)
```

该函数将字符数组 S1，S2，S3 的对应行级联起来（在水平方向上）。所有输入数组必须具有相同的行数（或者任何一个都是单串）。当输入是全字符数组时，输出也是一个字符数组；若任何输入是字符串的单元数组，则 strcat 返回一个字符串单元数组，该串是将 S1，S2，S3 的相应元素级联起来构成的。所有输入必须有同样的大小（也可以是标量）。任何输入也可以是字符数组，字符数组中尾部的空格被忽略，不出现在输出中。若输入是字符串单元数组，则不是这种情况。要保留这些尾部空格，就要用基于方括号的惯用的级联语法[S1 S2 S3 ...]。例如，

```
>> a = 'hello '   % Note the trailing blank space.
>> b = 'goodbye'
>> strcat(a, b)
ans =
    hellogoodbye
[a b]
ans =
    hello goodbye
```

函数 strvcat 的语法为

```
S = strvcat(t1, t2, t3, ...)
```

它形成包含文本串（或串矩阵）的字符数组 S，该数组的行的形式为 t1, t2, t3,…。要形成一个合法的矩阵，就必须对每个串添加空格。空白变量可以忽略。例如，若在前一个例子中应用字符串 a 和 b，则

```
>> strvcat(a, b)
ans =
    hello
    goodbye
```

函数 strcmp 使用语句

```
k = strcmp(str1, str2)
```

来比较变量中的两个字符串。若串相同，则返回 1（真）；否则，返回 0（假）。更通用的语法为

```
K = strcmp(S, T)
```

其中，S 或 T 是字符串的单元数组；若 S 的元素和 T 匹配，则 K 是一个包含 1 的数组（与 S 和 T 的大小相同）；若 S 的元素和 T 不匹配，则 K 是一个包含 0 的数组。S 和 T 的大小必须相同（或者可以是一个标量单元），每一个都可以是一个具有相同行数的字符数组。函数 strcmpi 与 strcmp 执行相同的操作，只不过忽略了字符的大小写。

函数 strncmp 的语法为

```
k = strncmp('str1', 'str2', n)
```

若字符串 str1 和 str2 中的前 n 个字符相同，则返回逻辑真(1)；否则，返回逻辑假(0)。参量 str1 和 str2 可以是字符串单元数组。语法

```
R = strncmp(S, T, n)
```

（其中，S与T可以是串的单元数组）在S和T的元素匹配（最多n个字符）时，将返回一个与S和T同样大小的包含1的数组R；否则，返回包含0的数组R。S与T必须具有相同的大小（或可以是一个标量单元）。每一个都可以是有正确行数的字符数组。命令strncmp是大小写敏感的。每个字符串开头和尾部的空格也包含在比较的范围之内。函数strncmpi与strncmp执行相同的操作，但不区分字符的大小写。

函数strfind使用语句：

```
I = strfind(str, pattern)
```

搜索字符串str来查找更短的字符串pattern，并以双精度数组I的形式返回每一个短串的开始索引。若在str中未发现pattern，或是pattern比str更长，则strfind返回一个空数组[]。

函数strjust的语法为

```
Q = strjust(A, direction)
```

其中，A是一个字符数组，direction有三种对齐方式，即'right'，'left'和'center'。默认的对齐方式是'right'。输出数组包含有与A相同的字符串，但根据指定的方向进行对齐。注意，字符的对齐意味着在串的开头和（或）尾部存在空格来为特定的操作提供空间。例如，用符号"□"表示一个空格字符，字符串'□□abc'的开头有两个空格字符，在'right'对齐方式下该字符串保持不变；在'left'对齐方式下变为'abc□□'，在'center'对齐方式下则变为'□abc□'。显然，对开头或结尾都没有空格的字符串，上述这些操作不会起任何作用。

函数strmatch使用语法

```
m = strmatch('str', STRS)
```

遍历字符数组或串的单元数组STRS的每一行，寻找以字符串str开头的字符串，并返回匹配行的索引。它的另一种语法

```
m = strmatch('str', STRS, 'exact')
```

仅返回与str精确匹配的STRS中的字符串的索引。例如，语句

```
>> m = strmatch('max', strvcat('max', 'minimax', 'maximum'));
```

返回m=[1;3]，因为由strvcat形成的数组的第1行和第3行以'max'开头。另一方面，语句

```
>> m = strmatch('max', strvcat('max', 'minimax', 'maximum'), 'exact');
```

返回m = 1，因为只有1行与'max'精确匹配。

函数strrep使用语法

```
r = strrep('str1', 'str2', 'str3')
```

将字符串str1中所有的字符串str2用字符串str3替换。若任何一个str1、str2或str3是一个串的单元数组，则该函数返回一个与str1、str2和str3同样大小的单元数组，这是用输入中的对应元素执行strrep得到的。所有输入的大小必须相同（或者可以是标量单元）。每个字符串也可以是一个具有正确行数的字符数组。例如，

```
>> s = 'Image processing and restoration.';
>> str = strrep(s, 'processing', 'enhancement')
str =
    Image enhancement and restoration.
```

产生

```
b =
    [1 2;3 4]
```

其中，b是一个含有9个字符的字符串，包括方括号、空格和一个分号。命令

```
>> eval(mat2str(A))
```

重新生成A。其他此类函数的解释与此类似。

表12.2中的最后一类函数处理基数的转换。例如，函数dec2base使用语句

```
str = dec2base(d, base)
```

将十进制整数转换为指定进制（base）的整数，其中d是一个小于2^52的非负整数，base是位于2与36之间的整数。返回的参量str是一个串。例如，下面的命令将23_{10}转换为进制为2的数，并以串的形式返回结果：

```
>> str = dec2base(23, 2)
str =
    10111
>> class(str)
ans =
    char
```

语句

```
str = dec2base(d, base, n)
```

产生一个至少为n位数字的表示。

12.4.2 串的匹配

除了表12.2中的串匹配和比较的函数外，相似性度量通常也很有用，这种度量很像12.2节中讨论的距离度量，我们将使用下面定义的度量来说明这种方法。

假定有两个区域边界a和b，它们已分别编码为两个字符串$a_1a_2\cdots a_m$和$b_1b_2\cdots b_n$。令α表示这两个字符串之间的匹配数，其中一个匹配在$a_k = b_k$时出现在第k个位置。不匹配的符号数为

$$\beta = \max(|a|,|b|) - \alpha$$

其中，|**参量**|是参量中字符串的长度（即符号的个数）。可以看出，当且仅当a和b是完全相同的字符串时，才有$\beta = 0$。

一种a与b之间相似性的简单度量是比值

$$R = \frac{\alpha}{\beta} = \frac{\alpha}{\max(|a|,|b|) - \alpha}$$

这一度量由Sze and Yang[1981]提出，该度量在完全匹配情况下为无穷大，若a和b中没一个相应的符号匹配，则其值为0（此时$\alpha = 0$）。

因为匹配是在相应的符号之间进行的，所以需要所有符号串以某种与位置无关的方式来“配准”，这样才能使前面讨论的方法有意义。配准两个符号串的一种方法是相对于另一个符号串移动一个串，直到R的值达到最大。这种或其他相似的匹配策略可以使用表12.2中详细列出的某些串操

作来实现。典型情况下，一种更有效的方法是，在提取串表示之前就在对边界的大小和方向进行归一化的基础上为所有符号串确定相同的起始点。这一方法将在例 12.3 中说明。

下面的 M 函数对两个字符串计算前面所述的相似性度量。

```
function R = strsimilarity(a, b)
%STRSIMILARITY Computes a similarity measure between two strings.
%   R = STRSIMILARITY(A, B) computes the similarity measure, R,
%   defined in Section 12.4.2 for strings A and B. The strings do not
%   have to be of the same length, but if one is shorter than other,
%   then it is assumed that the shorter string has been padded with
%   leading blanks so that it is brought into the necessary
%   registration prior to using this function. Only one of the
%   strings can have blanks, and these must be leading and/or
%   trailing blanks. Blanks are not counted when computing the length
%   of the strings for use in the similarity measure.

% Verify that a and b are character strings.
if ~ischar(a) | ~ischar(b)
   error('Inputs must be character strings.')
end

% Find any blank spaces.
I = find(a == ' ');
J = find(b == ' ');
LI = length(I); LJ = length(J);
if LI ~= 0 & LJ ~= 0
   error('Only one of the strings can contain blanks.')
end

% Pad the end of the appropriate string. It is assumed
% that they are registered in terms of their beginning
% positions.
a = a(:); b = b(:);
La = length(a); Lb = length(b);
if LI == 0 & LJ == 0
   if La > Lb
      b = [b; blanks(La - Lb)'];
   else
      a = [a; blanks(Lb - La)'];
   end
elseif isempty(I)
   Lb = length(b) - length(J);
   b = [b; blanks(La - Lb - LJ)'];
else
   La = length(a) - length(I);
   a = [a; blanks(Lb - La - LI)'];
end

% Compute the similarity measure.
I = find(a == b);
alpha = length(I);
den = max(La, Lb) - alpha;
if den == 0
   R = Inf;
else
   R = alpha/den;
end
```

这样就产生了一个其元素为1到8之间的数的串，1表示角度范围为$0° \leq \theta < 45°$，2表示角度范围为$45° \leq \theta < 90°$，依次类推，其中θ表示一个内角。

因为minperpoly输出的第一个顶点总是输入B边界左上角的顶点，所以s串的第一个元素与该顶点的内角相对应。这就自动将这些串配准了（若物体没有旋转），因为在所有图像中它们都是从左上角的顶点开始的。由minperpoly输出的顶点的方向是顺时针方向，因此s内的元素也是这个方向。最后用下面的命令将每个s从整数串转换为字符串：

```
>> s = int2str(s);
```

在这个例子中，对象都是大小可比的，且都竖直放置，因此，无论是对大小还是方向，都不需要归一化。若对象有任意的大小和方向，则可以使用在11.5节中讨论的特征向量变换的方法沿其主方向校准它们；然后，为进行归一化，可以使用11.4.1节中的围框得到对象的维数。

首先，我们使用函数strsimilarity来度量类别1中所有串之间的相似性。例如，使用下面的命令来计算类别1中的第一串与第二个串之间的相似性：

```
>> R = strsimilarity(s11, s12);
```

其中，第一个下标表示类别，第二个下标指明该类中的串号；用5个典型串得到的结果综合在表12.4中，其中，Inf表示无穷大（即早些时候讨论的最佳匹配）。表12.5显示了类别2中5个串的同类计算结果。表12.6显示了类别1与类别2的串之间的相似性度量值。注意，该表中的值要比前两个表中的值小得多，这说明这两类对象的R度量获得了较高的区别度。换言之，度量类内成员的相似性时，R的值相当大，这说明当与另一类的成员相比较时，匹配更加精确。

表12.4 类别1的串间的相似性度量R的值（所有显示的值都乘10）

R	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
s_{11}	Inf				
s_{12}	9.33	Inf			
s_{13}	26.25	12.31	Inf		
s_{14}	16.36	9.33	14.16	Inf	
s_{15}	22.22	14.17	14.01	19.02	Inf

表12.5 类别2的串间的相似性度量R的值（所有显示的值都乘10）

R	s_{21}	s_{22}	s_{23}	s_{24}	s_{25}
s_{21}	Inf				
s_{22}	10.00	Inf			
s_{23}	13.33	13.33	Inf		
s_{24}	7.50	13.31	18.00	Inf	
s_{25}	13.33	7.51	18.12	10.01	Inf

表12.6 类别1和类别2的串间的相似性度量R的值（所有显示的值都乘10）

R	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
s_{21}	2.03	0.01	1.15	1.17	0.75
s_{22}	1.15	1.61	1.16	0.75	2.07
s_{23}	2.08	1.15	2.08	2.06	2.08
s_{24}	1.60	1.62	1.59	1.14	2.61
s_{25}	1.61	0.36	0.74	1.60	1.16

小结

从第9章起，我们对图像处理的讨论开始从输出为图像的处理过程过渡到输出为图像特征的处理过程。尽管本章的内容是介绍性的，但所涵盖的主题对理解对象识别技术的现状是非常必要的。

学完这12章的内容后，读者应该能够掌握运用MATLAB和图像处理工具箱中的函数建立软件原型来解决图像处理问题的基础知识。更重要的是，本书的背景知识和所开发的大量新函数为如何拓展MATLAB和IPT的功能勾画出了基本蓝图。在给出大多数图像处理问题的特定特性后，充分理解本书所讲的内容会增加在宽泛的图像处理应用领域取得成功的机会。

附录A 函数汇总

前言

本附录的A.1节包含了图像处理工具箱中的所有函数以及前面几章中开发的所有新函数。较后的函数称为DIPUM函数，DIPUM一词源自本书标题的英文首字母。A.2节列出了全书所用到的MATLAB函数。清单中列出的节号指出了第一次使用和示例函数的位置。在某些情况下，我们给出了函数出现的多个位置，以说明函数的不同用法。一些IPT函数在我们的讨论中并未用到，此时会给出的只是这些函数的在线帮助而不是节号。A.2节中列出的所有MATLAB函数在本书中均已用到，且给出第一次使用它们的位置。

A.1 IPT和DIPUM函数

如下函数的分类方式类似于它们在IPT文档中的分类。当列出的函数在IPT函数中不存在时，会为其创建新的类别（如小波）。

函数类别和名称	描述	节号或其他位置
图像显示		
colorbar	显示彩条（MATLAB）	在线
getimage	由坐标轴得到图像数据	在线
ice(DIPUM)	交互彩色编辑	6.4
image	创建和显示图像对象（MATLAB）	在线
imagesc	缩放数据并显示为图像（MATLAB）	在线
immovie	由多帧图像制作电影	在线
imshow	显示图像	2.3
imview	在Image Viewer中显示图像	在线
montage	将多个图像帧显示为矩形蒙太奇	在线
movie	播放录制的电影帧（MATLAB）	在线
rgbcube(DIPUM)	显示一个彩色RGB立方体	6.1.1
subimage	在单个图形中显示多幅图像	在线
truesize	调整图像的显示尺寸	在线
warp	将图像显示为纹理映射的表面	在线
图像文件输入／输出		
dicominfo	从一条DICOM消息中读取元数据	在线
dicomread	读一幅DICOM图像	在线
dicomwrite	写一幅DICOM图像	在线
dicom-dict.txt	包含DICOM数据字典的文本文件	在线
dicomuid	产生DICOM惟一的识别器	在线
imfinfo	返回关于图像文件的信息（MATLAB）	2.4
imread	读图像文件（MATLAB）	2.2
imwrite	写图像文件（MATLAB）	2.4
图像算术		
imabsdiff	计算两幅图像的绝对差	2.10.2
imadd	两幅图像相加或把常数加到图像上	2.10.2

（续表）

函数类别和名称	描述	节号或其他位置
imcomplement	图像求补	2.10.2, 3.2.1
imdivide	两幅图像相除，或用常数除图像	2.10.2
imlincomb	计算图像的线性组合	2.10.2, 5.3.1
immultiply	两幅图像相乘或用常数乘图像	2.10.2
imsubtract	两幅图像相减，或从图像中减去常数	2.10.2
几何变换		
checkerboard	创建棋盘格图像	5.5
findbounds	求几何变换的输出范围	在线
fliptform	颠倒 TFORM 结构的输入/输出	在线
imcrop	修剪图像	在线
imresize	调整图像大小	在线
imrotate	旋转图像	11.4.3
imtransform	对图像应用几何变换	5.11.2
intline	整数坐标线绘制算法（未文档化的 IPT 函数）	2.10.2
makeresampler	创建重取样器结构	5.11.2
maketform	创建几何变换结构（TFORM）	5.11.1
pixeldup(DIPUM)	在两个方向上复制图像的像素	5.5
tformarray	对 N-D 数组应用几何变换	在线
tformfwd	应用正向几何变换	5.11.1
tforminv	应用反向几何变换	5.11.1
vistformfwd(DIPUM)	可视化正向几何变换	5.11.1
图像配准		
cpstruct2pairs	将 CPSTRUCT 转换为有效的控制点对	在线
cp2tform	由控制点对推断几何变换	5.11.3
cpcorr	使用互相关校准控制点位置	在线
cpselect	控制点选择工具	5.11.3
normxcorr2	归一化二维互相关	在线
像素值及统计		
corr2	计算二维相关系数	在线
covmatrix(DIPUM)	计算向量族的协方差矩阵	11.5
imcontour	创建图像数据的轮廓线	在线
imhist	显示图像数据的直方图	3.3.1
impixel	确定像素的彩色值	在线
improfile	计算沿着线段的像素值横截面	在线
mean2	计算矩阵元素的均值	3.2.3
pixval	显示关于像素的信息	2.3
regionprops	测量图像区域的属性	11.4.1
statmoments(DIPUM)	计算一幅图像直方图的统计中心矩	5.2.4
std2	计算矩阵元素的标准偏差	10.4.3
图像分析（包括分割、描述和识别）		
bayesgauss(DIPUM)	高斯模式的贝叶斯分类器	12.3.4
bound2eight(DIPUM)	将 4 连接边界转换为 8 连接边界	11.1.3
bound2four(DIPUM)	将 8 连接边界转换为 4 连接边界	11.1.3
bwboundaries	追踪区域边界	在线
bwtraceboundary	追踪单个边界	在线
bound2im(DIPUM)	将边界转换为图像	11.1.3
boundaries(DIPUM)	追踪区域边界	11.1.3
bsubsamp(DIPUM)	对边界二次取样	11.1.3
colorgrad(DIPUM)	计算一幅 RGB 图像的向量梯度	6.6.1

（续表）

函数类别和名称	描述	节号或其他位置
colorseg(DIPUM)	分割一幅彩色图像	6.6.2
connectpoly(DIPUM)	连接多边形的顶点	11.1.3
diameter(DIPUM)	测量图像区域的直径	11.3.2
edge	在一幅亮度图像中寻找边缘	10.1.3
fchcode(DIPUM)	计算边界的 Freeman 链码	11.2.1
frdescp(DIPUM)	计算傅里叶描绘子	11.3.3
graythresh	使用 Otsu 方法计算图像的全局阈值	10.3.1
hough(DIPUM)	Hough 变换	10.2
houghlines(DIPUM)	基于 Hough 变换提取线段	10.2.2
houghpeaks(DIPUM)	在 Hough 变换中检测峰值	10.2.1
houghpixels(DIPUM)	计算属于 Hough 变换 bin 的图像像素	10.2.2
ifrdescp(DIPUM)	计算逆傅里叶描绘子	11.3.3
imstack2vectors(DIPUM)	从图像堆栈提取向量	11.5
invmoments(DIPUM)	计算图像的不变矩	11.4.3
mahalanobis(DIPUM)	计算 Mahalanobis 距离	12.2
minperpoly(DIPUM)	计算最小周长多边形	11.2.2
polyangles(DIPUM)	计算多边形内角	12.4.2
princomp(DIPUM)	得到主分量向量和相关量	11.5
qtdecomp	执行四叉树分解	10.4.3
qtgetblk	得到四叉树分解中的块值	10.4.3
qtsetblk	在四叉树中设置块值	在线
randvertex(DIPUM)	随机置换多边形顶点	12.4.2
regiongrow(DIPUM)	由区域生长来执行分割	10.4.2
signature(DIPUM)	计算边界的标记	11.2.3
specxture(DIPUM)	计算图像的谱纹理	11.4.2
splitmerge(DIPUM)	使用分离-合并算法分割图像	10.4.3
statxture(DIPUM)	计算图像中纹理的统计度量	11.4.2
strsimilarity(DIPUM)	两个串间的相似性度量	12.4.2
x2majoraxis(DIPUM)	以区域的主轴排列坐标 x	11.3.2
图像压缩		
compare(DIPUM)	计算和显示两个矩阵间的误差	8.1
entropy(DIPUM)	计算矩阵的熵的一阶估计	8.2
huff2mat(DIPUM)	解码霍夫曼编码矩阵	8.2.3
huffman(DIPUM)	为符号源建立一个变长霍夫曼码	8.2.1
im2jpeg(DIPUM)	使用 JPEG 近似压缩一幅图像	8.5.1
im2jpeg2k(DIPUM)	使用 JPEG 2000 近似压缩一幅图像	8.5.2
imratio(DIPUM)	计算两幅图像或变量中的比特率	8.1
jpeg2im(DIPUM)	解码 IM2JPEG 压缩的图像	8.5.1
jpeg2k2im(DIPUM)	解码 IM2JPEG2K 压缩的图像	8.5.2
lpc2mat(DIPUM)	解压缩一维有损预测编码矩阵	8.3
mat2huff(DIPUM)	霍夫曼编码矩阵	8.2.2
mat2lpc(DIPUM)	使用一维有损预测编码压缩矩阵	8.3
quantize(DIPUM)	量化 UINT8 类矩阵的元素	8.4
图像增强		
adapthisteq	自适应直方图量化	在线
decorrstretch	对多通道图像应用去相关拉伸	在线
gscale(DIPUM)	按比例调整输入图像的亮度	3.2.3
histeq	使用直方图均衡化来增强对比度	3.3.2
intrans(DIPUM)	执行亮度变换	3.2.3

（续表）

函数类别和名称	描述	页号或其他位置
imadjust	调整图像亮度值或彩色映射	3.2.1
stretchlim	寻找对比度拉伸图像的限制	在线
图像噪声		
imnoise	给一幅图像添加噪声	3.5.2
imnoise2(DIPUM)	使用指定的 PDF 生成一个随机数数组	5.2.2
imnoise3(DIPUM)	生成周期噪声	5.2.3
线性和非线性空间滤波		
adpmedian(DIPUM)	执行自适应中值滤波	5.3.2
convmtx2	计算二维卷积矩阵	在线
dftcorr(DIPUM)	执行频率域相关	12.3.3
dftfilt(DIPUM)	执行频率域滤波	4.3.3
fspecial	创建预定义滤波器	3.5.1
medfilt2	执行二维中值滤波	3.5.2
imfilter	滤波二维和 N 维图像	3.4.1
ordfilt2	执行二维顺序统计滤波	3.5.2
spfilt(DIPUM)	执行线性和非线性空间滤波	5.3.1
wiener2	执行二维去噪滤波	在线
线性二维滤波器设计		
freqspace	确定二维频率响应间隔（MATLAB）	在线
freqz2	计算二维频率响应	4.4
fsamp2	使用频率取样设计二维 FIR 滤波器	在线
ftrans2	使用频率变换设计二维 FIR 滤波器	在线
fwind1	使用一维窗法设计二维滤波器	在线
fwind2	使用二维窗法设计二维滤波器	在线
hpfilter(DIPUM)	计算频率域高通滤波器	4.6.1
lpfilter(DIPUM)	计算频率域低通滤波器	4.5.2
图像去模糊（复原）		
deconvblind	使用盲去卷积去模糊图像	5.10
deconvlucy	使用 Lucy-Richardson 方法去模糊	5.9
deconvreg	使用规则化滤波器去模糊	5.8
deconvwnr	使用维纳滤波器去模糊	5.7
edgetaper	使用点扩散函数锐化边缘	5.7
otf2psf	光传递函数到点扩散函数	5.1
psf2otf	点扩散函数到光传递函数	5.1
图像变换		
dct2	二维离散余弦变换	8.5.1
dctmtx	离散余弦变换矩阵	8.5.1
fan2para	将扇形射束投影变换为并行射束	在线
fanbeam	计算扇形射束变换	在线
fft2	二维快速傅里叶变换（MATLAB）	4.2
fftn	N 维快速傅里叶变换（MATLAB）	在线
fftshift	颠倒 FFT 输出的象限（MATLAB）	4.2
idct2	二维逆离散余弦变换	在线
ifanbeam	计算扇形射束逆变换	在线
ifft2	二维快速傅里叶逆变换（MATLAB）	4.2
ifftn	N 维快速傅里叶逆变换（MATLAB）	在线
iradon	计算逆 Radon 变换	在线
para2fan	将并行射束投影变换为扇形射束	在线
phantom	生成头部仿真模型的图像	在线
radon	计算 Radon 变换	在线

（续表）

函数类别和名称	描述	页号或其他位置
小波		
wave2gray(DIPUM)	显示小波分解系数	7.3.2
waveback(DIPUM)	执行多灰度级二维快速小波逆变换	7.4
wavecopy(DIPUM)	存取小波分解结构的系数	7.3.1
wavecut(DIPUM)	在小波分解结构中置零系数	7.3.1
wavefast(DIPUM)	执行多灰度级二维快速小波变换	7.2.2
wavefilter(DIPUM)	构造小波分解和重构滤波器	7.2.2
wavepaste(DIPUM)	在小波分解结构中放置系数	7.3.1
wavework(DIPUM)	编辑小波分解结构	7.3.1
wavezero(DIPUM)	将小波细节系数设置为零	7.5
邻域和块处理		
bestblk	为块处理选择块大小	在线
blkproc	为图像实现不同的块处理	4.5.1
col2im	将矩阵列重排为块	4.5.1
colfilt	按列邻域操作	3.4.2
im2col	将图像块重排为列	8.5.1
nlfilter	执行一般的滑动邻域操作	3.4.2
形态学操作（亮度和二值图像）		
conndef	默认连通性	在线
imbothat	执行底帽滤波	9.6.2
imclearborder	抑制与图像边框相连的亮结构	9.6.1
imclose	关闭图像	9.3.1
imdilate	膨胀图像	9.2.1
imerode	腐蚀图像	9.2.4
imextendedmax	最大扩展变换	在线
imextendedmin	最小扩展变换	在线
imfill	填充图像区域和孔洞	9.5.3
imhmax	H最大变换	在线
imhmin	H最小变换	9.6.3
imimposemin	强制最小	10.5.3
imopen	打开图像	9.3.1
imreconstruct	形态学重构	9.5.1
imregionalmax	局部最大区域	在线
imregionalmin	局部最小区域	10.5.3
imtophat	执行顶帽滤波	9.6.7
watershed	分水岭变换	10.5.2
形态学操作（二值图像）		
applylut	使用查表法执行邻域操作	9.3.3
bwarea	计算二值图像中的对象面积	在线
bwareaopen	打开二值区域（删除小对象）	在线
bwdist	计算二值图像的距离变换	10.5.1
bweuler	计算二值图像的欧拉数	在线
bwhitmiss	二值击不中操作	9.3.2
bwlabel	在二维图像中标记连接分量	9.4
bwlabeln	在N维二值图像中标记连接分量	在线
bwmorph	对二值图像执行形态学操作	9.3.4
bwpack	打包二值图像	在线

（续表）

函数类别和名称	描述	页号或其他位置
数组操作		
circshift	循环地移位数组（MATLAB）	11.1.3
dftuv(DIPUM)	计算网格数组	4.5.1
padarray	填充数组	3.4.2
paddedsize(DIPUM)	计算用于FFT的最小填充尺寸	4.3.1
图像类型和类型转换		
changeclass	改变一幅图像的类（未文档化的IPT函数）	3.2.3
dither	使用抖动转换图像	6.1.3
gray2ind	将亮度图像转换为索引图像	6.1.3
grayslice	通过阈值处理从亮度图像创建索引图像	6.1.3
im2bw	通过阈值处理将图像转换为二值图像	2.7.2
im2double	将图像数组转换为双精度	2.7.2
im2java	将图像转换为Java图像（MATLAB）	在线
im2java2d	将图像转换为Java缓存的图像对象	在线
im2uint8	将图像数组转换为8比特无符号整数	2.7.2
im2uint16	将图像数组转换为16比特无符号整数	2.7.2
ind2gray	将索引图像转换为亮度图像	1.1.3
ind2rgb	将索引图像转换为RGB图像（MATLAB）	1.1.3
label2rgb	将标记矩阵转换为RGB图像	在线
mat2gray	将矩阵转换为亮度图像	2.7.2
rgb2gray	将RGB图像或彩色映射转换为灰度图像	6.1.3
rgb2ind	将RGB图像转换为索引图像	6.1.3
其他函数		
conwaylaws(DIPUM)	对单个像素应用Conway的遗传定律	9.3.3
manualhist(DIPUM)	交互地生成2模式直方图	3.3.3
twomodegauss(DIPUM)	生成一个2模式高斯函数	3.3.3
uintlut	基于查找表计算新数组值	在线
工具箱参数		
iptgetpref	获得图像处理工具箱参数的值	在线
iptsetpref	设置图像处理工具箱参数的值	在线

A.2 MATLAB函数

下面按字母顺序列出的MATLAB函数已在本书中用到。详细内容请参阅给出的节号或在线帮助。

MATLAB函数	说明	节号
A		
abs	绝对值和复数幅度	4.2
all	测试以确定所有的元素是否非零	2.10.2
ans	最新答案	2.10.2
any	测试任何非零元素	2.10.2
axis	轴缩放与显示	3.3.1
B		
bar	带状图	3.3.1
bin2dec	二进制至十进制数的转换	8.2.2
blanks	空串	12.4.1
break	终止执行for循环或while循环	2.10.3
C		
cart2pol	将笛卡儿坐标变换为极坐标或柱坐标	11.2.3

（续表）

MATLAB 函数	说明	节号
cat	级联数组	6.1.1
ceil	趋于无穷	4.2
cell	创建单元数组	8.2.1
celldisp	显示单元数组的内容	8.2.1, 11.1.1
cellfun	对单元数组中的每一个元素应用一个函数	11.1.1
cellplot	图形化地显示单元数组的结构	8.2.1
cellstr	从字符数组创建串的单元数组	12.4.1
char	创建字符数组（串）	2.10.5, 12.4.1
circshift	循环移位数组	11.1.3
colon	冒号操作符	2.8.1, 2.10.2
colormap	设置和得到当前的彩色映射	4.5.3, 6.1.3
computer	识别 MATLAB 正在其上运行的计算机的信息	2.10.2
continue	将控制传递给 for 或 while 循环的下一次迭代	2.10.3
conv2	二维卷积	7.2.2
ctranspose	向量和矩阵复数转置（实数转置见函数 transpose）	2.10.2
cumsum	累积求和	3.3.2
D		
dec2base	十进制数至基数的转换	12.4.2
dec2bin	十进制至二进制数的转换	8.2.2
diag	对角线矩阵和矩阵的对角线	6.6.2
diff	微分和近似导数	9.6.2
dir	显示目录列表	8.1
disp	显示文本或数组	2.10.5
double	转换为双精度	2.6.1
E		
edit	编辑或创建 M 文件	2.10.2
eig	寻找特征值和特征向量	11.5
end	终止 for, while, switch, try 和 if 语句，或指出最后的索引	2.8.1
eps	浮点相对精度	2.10.2, 3.2.2
error	显示错误信息	2.10.3
eval	执行一个包含 MATLAB 表达式的串	12.4.1
eye	单位矩阵	12.3.4
F		
false	创建错误数组，简记为 logical(0)	2.10.1, 10.4.2
feval	函数赋值	10.4.3
fft2	二维离散傅里叶变换	4.2
fftshift	将 DFT 的零频率分量移到谱的中心	4.2
fieldnames	返回一个结构的域名或一个对象的属性名	8.1
figure	创建一个图形对象	2.4
find	寻找非零元素的索引和值	5.2.2
fliplr	左右倒转矩阵	11.4.3
flipup	上下倒转矩阵	11.4.3
floor	趋于负无穷	4.2
for	以固定的次数重复一组语句	2.10.3
full	将稀疏矩阵转换为满矩阵	10.2
G		
gca	获得当前的轴句柄	3.3.1
get	获得目标属性	6.4
getfield	获得结构数组的域	B.2.2

(续表)

MATLAB函数	说明	节号
global	定义全局变量	8.2.1
grid	二维和三维图的栅格线	4.5.3
guidata	存储或检索应用数据	B.2.2
guide	启动 GUI Layout Editor	B.1
H		
help	在命令窗口显示MATLAB函数的帮助信息	2.10.1
hist	计算和/或显示直方图	5.2.3
histc	直方图计算	8.2.2
hold on	保持当前的图形和某些轴属性	3.3.2
I		
if	有条件地执行语句	2.10.3
ifft2	二维离散傅里叶逆变换	4.2
ifftshift	反向FFT移位	4.2
imag	复数的虚部	4.2
int16	转换为带符号的整数	2.5
inpolygon	检测多边形区域内部的点	11.2.2
input	请求用户输入	2.10.5
int2str	整数到串的转换	12.4.1
int32	转换为带符号的整数	2.5
int8	转换为带符号的整数	2.5
interp1q	快速一维线性插值	6.4
inv	计算逆矩阵	10.2.2
is*	见表2.9	2.10.2
iscellstr	确定某项是否是串的单元数组	2.10.2, 12.4.1
islogical	确定某项是否是逻辑数组	2.6.2
L		
ldivide	数组左除（矩阵左除见函数mldivide）	2.10.2
length	向量的长度	2.10.3
linspace	生成线性空间向量	2.8.2
load	从磁盘装载工作空间变量	8.3
log	自然对数	3.2.2
log10	以10为底的对数	3.2.2
log2	以2为底的对数	3.2.2
logical	将数值转换为逻辑值	2.6.2
lookfor	在所有帮助条目中搜索指定的关键字	2.10.2
lower	将字符串转换为小写	2.10.6
M		
magic	生成幻方矩阵（纵横图）	2.9
mat2str	将矩阵转换为串	12.4.1
max	数组的最大元素	2.10.2
mean	数组的平均值或均值	9.5
median	数组的中值	3.5.2
mesh	网格图	4.5.3
meshgrid	生成三维图形的X和Y矩阵	2.10.4
mfilename	当前运行的M文件名	B.2.1
min	数组的最小元素	2.10.2
minus	数组和矩阵相减	2.10.2
mldivide	矩阵左除（数组左除见函数ldivide）	2.10.2
mpower	矩阵的幂（数组的幂见函数power）	2.10.2

（续表）

MATLAB 函数	说明	节号
mrdivide	矩阵右除（数组右除见函数 rdivide）	2.10.2
mtimes	矩阵乘（数组乘见函数 times）	2.10.2
N		
nan or NaN	非数	2.10.2
nargchk	检查输入参量的数目	3.2.3
nargin	输入函数的参量数	3.2.3
nargout	输出函数的参量数	3.2.3
ndims	数组的维数	2.9
nextpow2	2的下一次幂	4.3.1
norm	向量和矩阵的范数	12.2
numel	数组中的元素数	2.10.3
O		
ones	生成全1的数组	2.9
P		
patch	创建斑纹图形对象	6.1.1
permute	重排多维数组的维数	12.2
persistent	定义持久变量	9.3.3
pi	圆的周长与其直径的比率	2.10.2
plot	线性二维图形	3.3.1
plus	数组和矩阵相加	2.10.2
pol2cart	将极坐标或圆柱坐标变换为笛卡儿坐标	11.2.3
pow2	2的幂和比例浮点数	8.2.2
power	数组的幂（矩阵的幂见函数 mpower）	2.10.2
print	打印至文件或硬拷贝设备	2.5
prod	数组元素的积	3.4.2
R		
rand	均匀分布的随机数和数组	2.9, 5.2.2
randn	正态分布随机数和数组	2.9, 5.2.2
rdivide	数组右除（矩阵右除见函数 mrdivide）	2.10.3
real	复数的实部	4.2
realmax	计算机所能表示的最大浮点数	2.10.2
realmin	计算机所能表示的最小浮点数	2.10.2
regexp	匹配正规表达式	12.4.1
regexpi	匹配正规表达式，忽略大小写	12.4.1
regexprep	用正规表达式代替串	12.4.1
rem	除后的余数	7.2.2
repmat	复制和排列数组	7.3.1
reshape	变形数组	8.2.2
return	返回调用函数	2.10.3
rot90	矩阵旋转 90°	3.4.1
round	四舍五入为最接近的整数	2.4
S		
save	将工作空间变量存入磁盘	8.2.3
set	设置对象属性	3.3.1
setfield	设置结构数组的域	B.2.3
shading	设置彩色阴影属性，本书中使用 interp 模式	4.5.3
sign	符号函数	8.5.2
single	转换为单精度	2.6.1
size	返回数组维数	2.2

（续表）

MATLAB 函数	说明	节号
sort	将元素按升序排列	8.2.1
sortrows	将行按升序排列	11.1.3
sparse	创建稀疏矩阵	10.2
spline	三次样条数据插值	6.4
sprintf	将格式化的数据写入字符串	2.10.3
stem	绘制离散序列数据	3.3.1
str*	字符串操作，见表 12.2	12.4.1
str2num	字符串到数的转换	2.10.5
strcat	字符串级联	12.4.1
strcmp	字符串比较	2.10.6, 12.4.1
strcmpi	字符串比较，忽略大小写	12.4.1
strfind	在另一个字符串中查找一个字符串	12.4.1
strjust	证明一个字符数组	12.4.1
strmatch	对一个字符串寻找可能的匹配	12.4.1
strncmp	比较两个字符串的前 n 个字符	12.4.1
strncmpi	比两个字符串的前 n 个字符，不计大小写	8.4, 12.4.1
strread	从字符串读取格式化数据	2.10.5
strrep	字符串查找和替换	12.4.1
strtok	字符串中的第一个标记	12.4.1
strvcat	字符串的垂向级联	12.4.1
subplot	将图形窗口细分为轴或子图的数组	7.2.1
sum	数组元素的和	2.8.2
surf	三维阴影表面图	4.5.3
switch	基于表达式在几种情况中切换	2.10.3
T		
text	创建文本对象	3.3.1
tic,toc	秒表计时器	2.10.4
times	数组乘（矩阵乘见函数 mtimes）	2.10.2
title	为当前的图形添加图题	3.3.1
transpose	矩阵或向量转置（复数据见函数 ctranspose）	2.8.1, 2.10.2
true	创建真数组，简写为 logical(1)	2.9, 10.4.2
try...catch	见表 2.11	2.10.3
U		
uicontrol	创建用户界面控制对象	B.2.1
uint16	转换为无符号整数	2.5
uint32	转换为无符号整数	2.5
uint8	转换为无符号整数	2.5
uiresume	控制程序执行	B.2.2
uiwait	控制程序执行	B.2.2
uminus	一元减	2.10.2
uplus	一元加	2.10.2
unique	向量的惟一元素	11.1.3
upper	将字符串转换为大写	2.10.6
V		
varargin	传递可变的参量数	3.2.3
vararout	返回可变的参量数	3.2.3
version	得到 MATLAB 版本数	2.10.2
view	观察点说明	4.5.3

附录B　ICE和MATLAB图形用户界面

前言

在本附录中，我们将开发在第6章介绍过的ice交互式彩色编辑（ICE）函数。本讨论假定读者熟悉了6.4节的相关内容。6.4节提供了许多使用ice的例子（例6.3至例6.7），如伪彩色和真彩色图像处理，并描述了ice的调用语法、输入参数和图形界面元素（它们都汇总在表6.4至表6.6中）。ice的功能是使用户交互地并图形化地生成彩色变换曲线，并在图像上实时或接近实时地显示产生的变换效果。

B.1　创建ICE的图形的用户界面

MATLAB的图形用户界面开发环境（GUIDE）为在M函数中集成图形用户界面（GUI）提供了一套丰富的工具。使用GUIDE，可以将(1)GUI布局（如其按钮、弹出菜单等）和(2) GUI操作的编程处理分成两个容易管理且相对独立的过程。由此产生的M函数构成了两个相同名称的文件（忽略扩展名）：

1. 一个带有扩展名.fig的文件（称为FIG文件），它包含了所有函数的GUI对象或元素的完整图形描述以及它们的空间排列。一个FIG文件包含有二进制数据，当相关的基于GUI的M函数执行时，这些二进制数据不需要分析。ICE的FIG文件(ice.fig)将在本节稍后描述。
2. 一个带有扩展名.m的文件（称为GUI M文件），它包含了控制GUI操作的代码。该文件包含有启动和退出GUI时调用的函数，以及当用户与GUI对象交互时（如按下一个按钮时）执行的回调函数。ICE的GUI M文件（ice.m）将在下一节描述。

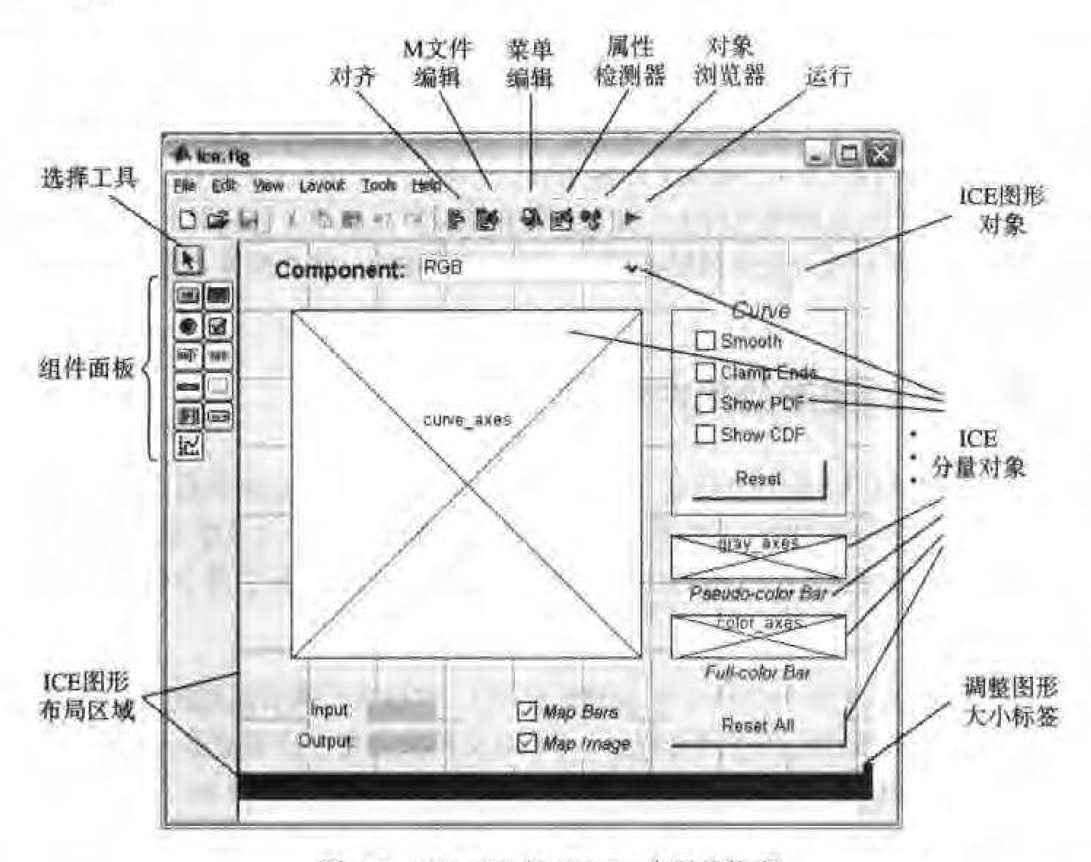
为了从MATLAB命令窗启动GUIDE，可键入

```
guide filename
```

其中，filename是当前路径中已存在的一个FIG文件名。若省略filename，则GUIDE将打开一个新窗口（即空白窗口）。

图B.1显示了适用于交互式彩色编辑器（ICE）布局的GUIDE布局编辑器（在MATLAB提示符>>后输入ice 来启动）。在开发用户界面的实体模型时，布局编辑器用于选择、放置、排列以及操作图形对象。左侧的按钮形成了组件面板，它包含有支持的GUI对象，如按式按钮、切换式按钮、单选按钮、复选框、编辑文字、静态文字、滑动条、帧、列表框、弹出菜单和轴。每一个对象的操作方式均类似于标准的Windows对象。任意对象的组合都可以加到布局编辑器右侧布局区中的图形对象上。注意，ICE GUI包含有复选框（Smooth, Clamp Ends, Show PDF, Show CDF, Map Bars和Map Image）、静态文字（“Component:”，“Curve”等）、描绘曲线控件—一个区域、

两个按钮（Reset和Reset All）、一个选择颜色变换曲线的弹出菜单、用于显示所选曲线（利用相关的控制点）的三个axes对象等。组成ICE元素的分层列表（点击布局编辑器上部任务栏中的Object Browser按钮得到）如图B.2(a)所示。注意，每一个元素都有惟一的名称或标签。例如，用于曲线显示的轴对象（在列表的上方）被赋予标识符curve_axes［在图B.2(a)中，标识符是左括号后的第一项］。



图B.1 ICE GUI的GUIDE布局编辑器

标签是所有GUI对象的常见属性之一。表征特定对象属性的滚动列表，可以通过选择对象［在图B.2(a)所示的Object Browser列表中或在图B.1所示的布局区域中使用Selection Tool（选择工具）］或点击布局编辑器任务栏上的Property Inspector（属性检测器）按钮显示出来。图B.2(b)显示了选择图B.2(a)中的figure对象时产生的列表。注意，figure对象的Tag属性［在图B.2(b)中突出显示的部分］是ice。这一属性很重要，因为GUIDE使用它自动产生figure回调函数名。例如，当使用鼠标按钮在图形窗口上方点击时，在滚动的Property Inspector窗口底部的WindowButtonDownFcn属性将被分配一个名称ice_WindowButtonDownFcn。回忆可知，回调函数仅是当用户与GUI对象交互时所执行的M函数。其他值得注意的属性（通常是所有GUI对象的通用属性）包括Position和Units属性，它们定义了对象的大小和位置。

最后，我们注意到特殊对象的某些属性是惟一的。例如，一个按钮对象有Callback特性，它定义了按下按钮时所执行的函数，而String属性则确定了按钮的标记。ICE Reset按钮的Callback属性是reset_pushbutton_Callback［注意在回调函数名称集成了来自图B.2(a)的Tag属性］；其String属性是“Reset”。但要注意的是，Reset按钮没有WindowButtonMotionFcn属性；它是“图形”对象所特有的属性。

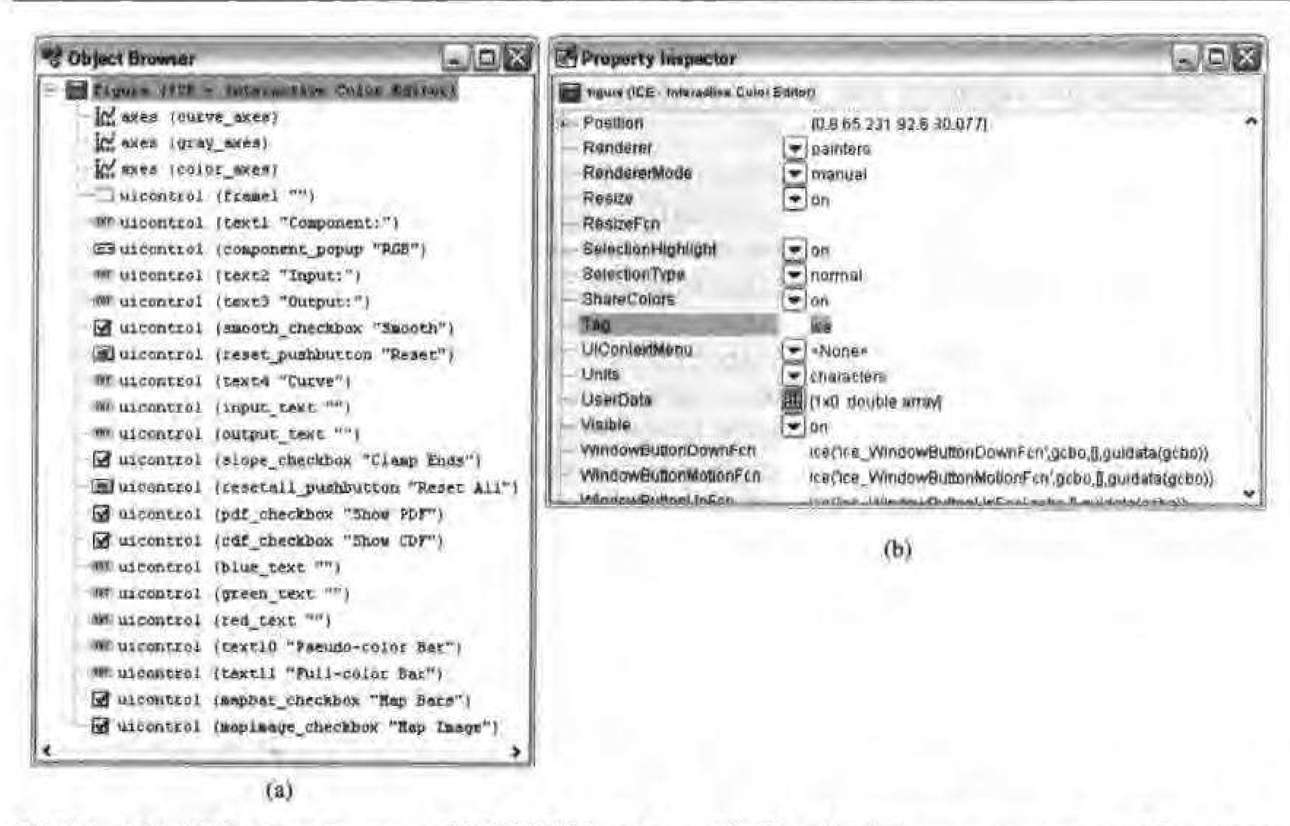


图 B.2 (a)GUIDE Object Browser（对象浏览器）; (b) ICE “图形”对象的 Property Inspector（属性检测器）

B.2 ICE 界面编程

在第一次存储 ICE FIG 文件或第一次运行 GUI 时［如在布局编辑器的任务栏上点击 Run（运行）按钮］，GUIDE 会生成一个称为 `ice.m` 的初始 GUI M 文件。这个可用标准文本编辑器或 MATLAB M文件编辑器修改的文件确定了界面响应用户动作的方式。为ICE自动生成的GUI M文件如下所示：

```
function varargout = ice(varargin)
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',          mfilename, ...
                   'gui_Singleton',     gui_Singleton, ...
                   'gui_OpeningFcn',    @ice_OpeningFcn, ...
                   'gui_OutputFcn',     @ice_OutputFcn, ...
                   'gui_LayoutFcn',     [], ...
                   'gui_Callback',      []);
if nargin & ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
function ice_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
```

```
% uiwait(handles.figure1);

function varargout = ice_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;

function ice_WindowButtonDownFcn(hObject, eventdata, handles)
function ice_WindowButtonMotionFcn(hObject, eventdata, handles)
function ice_WindowButtonUpFcn(hObject, eventdata, handles)
function component_popup_Callback(hObject, eventdata, handles)
function smooth_checkbox_Callback(hObject, eventdata, handles)
function reset_pushbutton_Callback(hObject, eventdata, handles)
function slope_checkbox_Callback(hObject, eventdata, handles)
function resetall_pushbutton_Callback(hObject, eventdata, handles)
function pdf_checkbox_Callback(hObject, eventdata, handles)
function cdf_checkbox_Callback(hObject, eventdata, handles)
function mapbar_checkbox_Callback(hObject, eventdata, handles)
function mapimage_checkbox_Callback(hObject, eventdata, handles)
```

这个自动生成的文件是开发功能完整的 ice 界面的有用起点或原型（注意，为节省空间，我们去除了许多 GUIDE 产生的注释文件）。在下面几节中，我们会将代码分为四个基本部分：(1) 两个“DO NOT EDIT”注释行间的初始化代码；(2)图形的打开和输出函数（ice_OpeningFcn 和 ice_OutputFcn）；(3)图形回调函数（如函数 ice_WindowButtonDownFcn，ice_WindowButtonMotionFcn 和 ice_WindowButtonUpFcn）；(4)对象回调函数（如 reset_pushbutton_Callback）。在考虑每一部分时，我们都将给出相应小节所包含的 ice 函数的完整开发版本，并且讨论将集中在多数 GUI M 文件开发者感兴趣的特征上。在每一节中介绍的代码将不再组合为单个完整的 ice.m 程序清单。

我们已在6.4节中描述过 ice 的操作。在下面的帮助文本块中，我们还将从全部开发 ice.m M 函数的角度对其进行总结：

```
%ICE Interactive Color Editor.
%
%   OUT = ICE('Property Name', 'Property Value', ...) transforms an
%   image's color components based on interactively specified mapping
%   functions. Inputs are Property Name/Property Value pairs:
%
%     Name               Value
%     --------------     ---------------------------------------------
%     'image'            An RGB or monochrome input image to be
%                        transformed by interactively specified
%                        mappings.
%     'space'            The color space of the components to be
%                        modified. Possible values are 'rgb', 'cmy',
%                        'hsi', 'hsv', 'ntsc' (or 'yiq'), 'ycbcr'. When
%                        omitted, the RGB color space is assumed.
%     'wait'             If 'on' (the default), OUT is the mapped input
%                        image and ICE returns to the calling function
%                        or workspace when closed. If 'off', OUT is the
%                        handle of the mapped input image and ICE
%                        returns immediately.
%
%   EXAMPLES:
```

B.2.1 初始化代码

开始GUI M文件（在B.2节的开始部分）中代码的打开部分是初始化代码的一个标准GUIDE生成块。其目的是用M文件的伴随FIG文件（见B.1节）建立和显示ICE的GUI，并且控制对所有内部M文件函数的存取。正像包含“DO NOT EDIT”的注释行所指出的那样，初始化代码不应修改。每次调用ice时，初始化块将建立一个调用gui_State的结构，该结构包含有存取ice函数的信息。例如，命名的域gui_Name（如gui_State.gui_Name）包含MATLAB函数mfilename，它返回当前执行的M文件的名称。采用类似的方式，使用GUIDE生成的ice的打开和输出函数的名称加载域gui_OpeningFcn和gui_OutputFcn（将在下节讨论）。若用户激活了一个ICE GUI对象（如点击一个按钮），则对象的回调函数名称会添加为域gui_Callback［回调函数的名称将在varargin(1)中以字符串的形式传递］。

在形成gui_State结构后，它将与varargin(:)一起作为输入参量传递给函数gui_mainfcn。这个MATLAB函数处理GUI的创建、布局和回调。对于ice，该函数建立和显示用户界面，并产生所有需要调用的打开、输出和回调函数。因为MATLAB的旧版本中可能不包含这些函数，所以通过从**File**菜单中选择**Export...**，可生成标准GUI M文件的单机版本（即无FIG文件的版本）。在单机版本中，函数gui_mainfcn与两个支持子程序ice_LayoutFcn和local_openfig附加到了由M文件确定的标准FIG文件中。ice_LayoutFcn的作用是创建ICE GUI。在ice的单机版本中，它由语句

```
h1 = figure(...
'Units', 'characters',...
'Color', [0.87843137254902 0.874509803921569 0.890196078431373],...
'Colormap', [0 0 0.5625;0 0 0.625;0 0 0.6875;0 0 0.75;...
            0 0 0.8125;0 0 0.875;0 0 0.9375;0 0 1;0 0.0625 1;...
            0 0.125 1;0 0.1875 1;0 0.25 1;0 0.3125 1;0 0.375 1;...
            0 0.4375 1;0 0.5 1;0 0.5625 1;0 0.625 1;0 0.6875 1;...
            0 0.75 1;0 0.8125 1;0 0.875 1;0 0.9375 1;0 1 1;...
            0.0625 1 1;0.125 1 0.9375;0.1875 1 0.875;...
            0.25 1 0.8125;0.3125 1 0.75;0.375 1 0.6875;...
            0.4375 1 0.625;0.5 1 0.5625;0.5625 1 0.5;...
            0.625 1 0.4375;0.6875 1 0.375;0.75 1 0.3125;...
            0.8125 1 0.25;0.875 1 0.1875;0.9375 1 0.125;...
            1 1 0.0625;1 1 0;1 0.9375 0;1 0.875 0;1 0.8125 0;...
            1 0.75 0;1 0.6875 0;1 0.625 0;1 0.5625 0;1 0.5 0;...
            1 0.4375 0;1 0.375 0;1 0.3125 0;1 0.25 0;...
            1 0.1875 0;1 0.125 0;1 0.0625 0;1 0 0;0.9375 0 0;...
            0.875 0 0;0.8125 0 0;0.75 0 0;0.6875 0 0;0.625 0 0;...
            0.5625 0 0],...
'IntegerHandle', 'off',...
'InvertHardcopy', get(0, 'defaultfigureInvertHardcopy'),...
'MenuBar', 'none',...
'Name', 'ICE - Interactive Color Editor',...
'NumberTitle', 'off',...
'PaperPosition', get(0, 'defaultfigurePaperPosition'),...
'Position', [0.8 65.2307692307693 92.6 30.0769230769231],...
'Renderer', get(0, 'defaultfigureRenderer'),...
'RendererMode', 'manual',...
'WindowButtonDownFcn', 'ice(''ice_WindowButtonDownFcn'', gcbo, [],...
```

```
handles.cindex = 1;                     % Index of selected curve
handles.node = 0;                       % Index of selected control point
handles.below = 1;                      % Index of node below control point
handles.above = 2;                      % Index of node above control point
handles.smooth = [0; 0; 0; 0];          % Curve smoothing states
handles.slope = [0; 0; 0; 0];           % Curve end slope control states
handles.cdf = [0; 0; 0; 0];             % Curve CDF states
handles.pdf = [0; 0; 0; 0];             % Curve PDF states
handles.output = [];                    % Output image handle
handles.df = [];                        % Input PDFs and CDFs
handles.colortype = 'rgb';              % Input image color space
handles.input = [];                     % Input image data
handles.imagemap = 1;                   % Image map enable
handles.barmap = 1;                     % Bar map enable
handles.graybar = [];                   % Pseudo (gray) bar image
handles.colorbar = [];                  % Color (hue) bar image

% Process Property Name/Property Value input argument pairs.
wait = 'on';
if (nargin > 3)
   for i = 1:2:(nargin - 3)
      if nargin - 3 == i
         break;
      end
      switch lower(varargin{i})
      case 'image'
         if ndims(varargin{i + 1}) == 3
            handles.input = varargin{i + 1};
         elseif ndims(varargin{i + 1}) == 2
            handles.input = cat(3, varargin{i + 1}, ...
                                varargin{i + 1}, varargin{i + 1});
         end
         handles.input = double(handles.input);
         inputmax = max(handles.input(:));
         if inputmax > 255
            handles.input = handles.input / 65535;
         elseif inputmax > 1
            handles.input = handles.input / 255;
         end

      case 'space'
         handles.colortype = lower(varargin{i + 1});
         switch handles.colortype
         case 'cmy'
            list = {'CMY' 'Cyan' 'Magenta' 'Yellow'};
         case {'ntsc', 'yiq'}
            list = {'YIQ' 'Luminance' 'Hue' 'Saturation'};
            handles.colortype = 'ntsc';
         case 'ycbcr'
            list = {'YCbCr' 'Luminance' 'Blue' ...
                    'Difference' 'Red Difference'};
         case 'hsv'
```

```
            list = {'HSV' 'Hue' 'Saturation' 'Value'};
         case 'hsi'
            list = {'HSI' 'Hue' 'Saturation' 'Intensity'};
         otherwise
            list = {'RGB' 'Red' 'Green' 'Blue'};
            handles.colortype = 'rgb';
         end
         set(handles.component_popup, 'String', list);
      case 'wait'
         wait = lower(varargin{i + 1});
      end
   end
end

% Create pseudo- and full-color mapping bars (grays and hues). Store
% a color space converted 1x128x3 line of each bar for mapping.
xi = 0:1/127:1;      x = 0:1/6:1;      x = x';
y = [1 1 0 0 0 1 1; 0 1 1 1 0 0 0; 0 0 0 1 1 1 0]';
gb = repmat(xi, [1 1 3]);        cb = interp1q(x, y, xi');
cb = reshape(cb, [1 128 3]);
if ~strcmp(handles.colortype, 'rgb')
   gb = eval(['rgb2' handles.colortype '(gb)']);
   cb = eval(['rgb2' handles.colortype '(cb)']);
end
gb = round(255 * gb);      gb = max(0, gb);      gb = min(255, gb);
cb = round(255 * cb);      cb = max(0, cb);      cb = min(255, cb);
handles.graybar = gb;      handles.colorbar = cb;

% Do color space transforms, clamp to [0, 255], compute histograms
% and cumulative distribution functions, and create output figure.
if size(handles.input, 1)
   if ~strcmp(handles.colortype, 'rgb')
      handles.input = eval(['rgb2' handles.colortype ...
                            '(handles.input)']);
   end
   handles.input = round(255 * handles.input);
   handles.input = max(0, handles.input);
   handles.input = min(255, handles.input);
   for i = 1:3
      color = handles.input(:, :, i);
      df = hist(color(:), 0:255);
      handles.df = [handles.df; df / max(df(:))];
      df = df / sum(df(:));   df = cumsum(df);
      handles.df = [handles.df; df];
   end
   figure;     handles.output = gcf;
end

% Compute ICE's screen position and display image/graph.
set(0, 'Units', 'pixels');      ssz = get(0, 'Screensize');
set(handles.ice, 'Units', 'pixels');
uisz = get(handles.ice, 'Position');
if size(handles.input, 1)
```

```
      fsz = get(handles.output, 'Position');
      bc = (fsz(4) - uisz(4)) / 3;
      if bc > 0
         bc = bc + fsz(2);
      else
         bc = fsz(2) + fsz(4) - uisz(4) - 10;
      end
      lc = fsz(1) + (size(handles.input, 2) / 4) + (3 * fsz(3) / 4);
      lc = min(lc, ssz(3) - uisz(3) - 10);
      set(handles.ice, 'Position', [lc bc 463 391]);
   else
      bc = round((ssz(4) - uisz(4)) / 2) - 10;
      lc = round((ssz(3) - uisz(3)) / 2) - 10;
      set(handles.ice, 'Position', [lc bc uisz(3) uisz(4)]);
   end
   set(handles.ice, 'Units', 'normalized');
   graph(handles);     render(handles);
   % Update handles and make ICE wait before exit if required.
   guidata(hObject, handles);
   if strcmpi(wait, 'on')
      uiwait(handles.ice);
   end

   %-------------------------------------------------------------------------%
   function varargout = ice_OutputFcn(hObject, eventdata, handles)
   %   After ICE is closed, get the image data of the current figure
   %   for the output. If 'handles' exists, ICE isn't closed (there was
   %   no 'uiwait') so output figure handle.

   if max(size(handles)) == 0
      figh = get(gcf);
      imageh = get(figh.Children);
      if max(size(imageh)) > 0
         image = get(imageh.Children);
         varargout{1} = image.CData;
      end
   else
      varargout{1} = hObject;
   end
```

这里，我们不再关注这些函数的复杂细节（关于特定函数的索引或帮助，请读者查阅代码的注释和附录A），而只关注多数GUI打开和输出函数的如下共性：

1. handles结构在多数GUI M文件中扮演了重要角色。它服务于两个至关重要的函数。因为handles结构为界面中的所有图形对象提供了句柄，所以可用于访问和修改对象属性。例如，ice打开函数使用

```
set(handles.ice, 'Units', 'pixels');
uisz = get(handles.ice, 'Position');
```

来访问ICE GUI的大小和位置（以像素为单位）。这可通过设置ice图形的Units属性来完成。这些句柄以handles.ice的形式用于'pixels'，然后读取图形的Position属性（使用get函数）。返回图形对象属性值的get函数也可用于获得计算机的显示区域，方

法是通过打开函数末尾的ssz = get(0,'Screensize')语句。其中，0是计算机显示的句柄（即根图形），'Screensize'是包含其宽度的属性。

除了提供访问GUI对象的功能之外，handles结构也是共享应用数据的有用通道。注意，handles结构对23个全局ice参数（其分布从handles.updown中的鼠标状态到handles.input中的全部输入图像）都保留默认值。这些参数在ice的每一次调用都必须保留，且要添加到位于ice_OpeningFcn的起始处的handles中。例如，handles.set1是由下列语句创建的全局参数：

```
handles.set1 = [0 0; 1 1]
```

其中，set1是一个命令的域，它包含有将添加到handles结构的彩色映射函数的控制点，且[0 0; 1 1]是其默认值[曲线端点(0, 0)和(1, 1)]。在退出函数以前，函数中的handles已被修改，必须调用

```
guidata(hObject, handles)
```

来将变量handles存储为带有句柄hObject的图形的应用数据。

2. 像许多内建图形函数那样，ice_OpeningFcn以属性名称和数值对的形式来处理输入参量（除hObject, eventdata和handles外）。当有三个以上的输入参量时（换言之，若nargin > 3），则执行跳过成对输入参量的循环[for i = 1:2:(nargin-3)]。对于每一对输入，第一对输入用于驱动switch结构，

```
switch lower(varargin{i})
```

它适当地处理第二个参数。例如，对于case'space'，语句

```
handles.colortype = lower(varargin{i + 1});
```

为输入对的第二个参量设置命名域colortype。然后，该值用于设置ICE的彩色分量弹出选项（即对象component_popup的String属性）。稍后，它将用于把输入图像的分量变换为期望的映射空间，语句如下：

```
handles.input = eval(['rgb2' ...
    handles.colortype '(handles.input)']);
```

其中，内建函数eval(s)会导致MATLAB像表达式或语句那样执行串s（关于函数eval的详细信息，请参阅12.4.1节）。若handles.input是'hsv'，则eval参量['rgb2' 'hsv' '(handles.input)']将变成级联串'rgb2hsv(handles.input)'，它将作为一个标准的MATLAB表达式来执行，其作用是将输入图像的RGB分量变换到HSV彩色空间（见6.2.3节）。

3. 语句

```
% uiwait(handles.figure1);
```

在起始GUI M文件中被转换为ice_OpeningFcn的最终版本中的条件语句

```
if strcmpi(wait, 'on') uiwait(handles.ice); end
```

一般来说，

```
uiwait(fig)
```

会在执行uiresume或毁坏图形fig（如关闭）后，终止MATLAB代码流的执行。若无输入参量，则uiwait与uiwait(gcf)相同，这时MATLAB函数gcf会返回当前图的句柄。

```
            dfx, handles.df(i + 5, :), 'b-', ...
            nodes(:, 1), nodes(:, 2), 'k-', ...
            nodes(:, 1), nodes(:, 2), 'ko', ...
            'Parent', handles.curve_axes);
    end

% Do the same for smooth (cubic spline) interpolations.
else
   x = 0:0.01:1;

   if ~handles.slope(handles.cindex)
      y = spline(nodes(:, 1), nodes(:, 2), x);
   else
      y = spline(nodes(:, 1), [ 0; nodes(:, 2); 0] , x);
   end
   i = find(y > 1);       y(i) = 1;
   i = find(y < 0);        y(i) = 0;
   if (~handles.pdf(c) & ~handles.cdf(c)) | ...
         (size(handles.df, 2) == 0)
      plot(nodes(:, 1), nodes(:, 2), 'ko', x, y, 'b-', ...
            'Parent', handles.curve_axes);
   elseif c > 1
      i = 2 * c - 2 - handles.pdf(c);
      plot(dfx, handles.df(i, :), [ colors(c) '-'], ...
            nodes(:, 1), nodes(:, 2), 'ko', x, y, 'k-', ...
            'Parent', handles.curve_axes);
   elseif c == 1
      i = handles.cdf(c);
      plot(dfx, handles.df(i + 1, :), 'r-', ...
            dfx, handles.df(i + 3, :), 'g-', ...
            dfx, handles.df(i + 5, :), 'b-', ...
            nodes(:, 1), nodes(:, 2), 'ko', x, y, 'k-', ...
            'Parent', handles.curve_axes);
  end
end

% Put legend if more than two curves are shown.
s = handles.colortype;
if strcmp(s, 'ntsc')
   s = 'yiq';
end
if (c == 1) & (handles.pdf(c) | handles.cdf(c))
   s1 = [ '-- ' upper(s(1))];
   if length(s) == 3
      s2 = [ '-- ' upper(s(2))];         s3 = [ '-- ' upper(s(3))];
   else
      s2 = [ '-- ' upper(s(2)) s(3)];    s3 = [ '-- ' upper(s(4)) s(5)];
   end
else
   s1 = '';       s2 = '';       s3 = '';
end
set(handles.red_text,  'String', s1);
set(handles.green_text, 'String', s2);
```

```
set(handles.blue_text, 'String', s3);
%-------------------------------------------------------------------------%
function [inplot, x, y] = cursor(h, handles)
%   Translate the mouse position to a coordinate with respect to
%   the current plot area, check for the mouse in the area and if so
%   save the location and write the coordinates below the plot.

set(h, 'Units', 'pixels');
p = get(h, 'CurrentPoint');
x = (p(1, 1) - handles.plotbox(1)) / handles.plotbox(3);
y = (p(1, 2) - handles.plotbox(2)) / handles.plotbox(4);
if x > 1.05 | x < -0.05 | y > 1.05 | y < -0.05
   inplot = 0;
else
   x = min(x, 1);      x = max(x, 0);
   y = min(y, 1);      y = max(y, 0);
   nodes = getfield(handles, handles.curve);
   x = round(256 * x) / 256;
   inplot = 1;
   set(handles.input_text, 'String', num2str(x, 3));
   set(handles.output_text, 'String', num2str(y, 3));
end
set(h, 'Units', 'normalized');
%-------------------------------------------------------------------------%
function y = render(handles)
%   Map the input image and bar components and convert them to RGB
%   (if needed) and display.

set(handles.ice, 'Interruptible', 'off');
set(handles.ice, 'Pointer', 'watch');
ygb = handles.graybar;       ycb = handles.colorbar;
yi = handles.input;          mapon = handles.barmap;
imageon = handles.imagemap & size(handles.input, 1);

for i = 2:4
   nodes = getfield(handles, ['set' num2str(i)]);
   t = lut(nodes, handles.smooth(i), handles.slope(i));
   if imageon
      yi(:, :, i - 1) = t(yi(:, :, i - 1) + 1);
   end
   if mapon
      ygb(:, :, i - 1) = t(ygb(:, :, i - 1) + 1);
      ycb(:, :, i - 1) = t(ycb(:, :, i - 1) + 1);
   end
end
t = lut(handles.set1, handles.smooth(1), handles.slope(1));
if imageon
   yi = t(yi + 1);
end
if mapon
   ygb = t(ygb + 1);      ycb = t(ycb + 1);
end
```

```
if ~strcmp(handles.colortype, 'rgb')
   if size(handles.input, 1)
      yi = yi / 255;
      yi = eval([ handles.colortype '2rgb(yi)']);
      yi = uint8(255 * yi);
   end
   ygb = ygb / 255;      ycb = ycb / 255;
   ygb = eval([ handles.colortype '2rgb(ygb)']);
   ycb = eval([ handles.colortype '2rgb(ycb)']);
   ygb = uint8(255 * ygb);      ycb = uint8(255 * ycb);
else
   yi = uint8(yi);      ygb = uint8(ygb);      ycb = uint8(ycb);
end

if size(handles.input, 1)
   figure(handles.output);      imshow(yi);
end
ygb = repmat(ygb, [ 32 1 1]);      ycb = repmat(ycb, [ 32 1 1]);
axes(handles.gray_axes);           imshow(ygb);
axes(handles.color_axes);          imshow(ycb);
figure(handles.ice);
set(handles.ice, 'Pointer', 'arrow');
set(handles.ice, 'Interruptible', 'on');

%-------------------------------------------------------------------------%
function t = lut(nodes, smooth, slope)
%  Create a 256 element mapping function from a set of control
%  points. The output values are integers in the interval [ 0, 255].
%  Use piecewise linear or cubic spline with or without zero end
%  slope interpolation.

t = 255 * nodes;      i = 0:255;
if ~smooth
   t = [ t; 256 256];    t = interp1q(t(:, 1), t(:, 2), i');
else
   if ~slope
      t = spline(t(:, 1), t(:, 2), i);
   else
      t = spline(t(:, 1), [ 0; t(:, 2); 0], i);
   end
end
t = round(t);      t = max(0, t);      t = min(255, t);

%-------------------------------------------------------------------------%
function out = spreadout(in)
% Make all x values unique.

% Scan forward for non-unique x's and bump the higher indexed x--
% but don't exceed 1. Scan the entire range.
nudge = 1 / 256;
for i = 2:size(in, 1) - 1
   if in(i, 1) <= in(i - 1, 1)
      in(i, 1) = min(in(i - 1, 1) + nudge, 1);
   end
```

```
    else
       node = below;
    end
    deletednode = 0;

    switch get(hObject, 'SelectionType')
    case 'normal'
       if node == above
          above = min(above + 1, size(nodes, 1));
       elseif node == below
          below = max(below - 1, 1);
       end
       if node == size(nodes, 1)
          below = above;
       elseif node == 1
          above = below;
       end
       if x > nodes(above, 1)
          x = nodes(above, 1);
       elseif x < nodes(below, 1)
          x = nodes(below, 1);
       end
       handles.node = node;     handles.updown = 'down';
       handles.below = below;   handles.above = above;
       nodes(node, :) = [x y];
    case 'extend'
       if ~length(find(nodes(:, 1) == x))
          nodes = [nodes(1:below, :); [x y]; nodes(above:end, :)];
          handles.node = above;     handles.updown = 'down';
          handles.below = below;   handles.above = above + 1;
       end
    case 'alt'
       if (node ~= 1) & (node ~= size(nodes, 1))
          nodes(node, :) = [];   deletednode = 1;
       end
       handles.node = 0;
       set(handles.input_text, 'String', '');
       set(handles.output_text, 'String', '');
    end

    handles = setfield(handles, handles.curve, nodes);
    guidata(hObject, handles);
    graph(handles);
    if deletednode
       render(handles);
    end
end

%-------------------------------------------------------------------------%
function ice_WindowButtonMotionFcn(hObject, eventdata, handles)
%   Do nothing unless a mouse 'down' event has occurred. If it has,
%   modify control point and make new mapping function.
```

```
if ~strcmpi(handles.updown, 'down')
   return;
end
[inplot, x, y] = cursor(hObject, handles);
if inplot
   nodes = getfield(handles, handles.curve);
   nudge = handles.smooth(handles.cindex) / 256;
   if (handles.node ~= 1) & (handles.node ~= size(nodes, 1))
      if x >= nodes(handles.above, 1)
         x = nodes(handles.above, 1) - nudge;
      elseif x <= nodes(handles.below, 1)
         x = nodes(handles.below, 1) + nudge;
      end
   else
      if x > nodes(handles.above, 1)
         x = nodes(handles.above, 1);
      elseif x < nodes(handles.below, 1)
         x = nodes(handles.below, 1);
      end
   end
   nodes(handles.node, :) = [x y];
   handles = setfield(handles, handles.curve, nodes);
   guidata(hObject, handles);
   graph(handles);
end
%-------------------------------------------------------------------------%
function ice_WindowButtonUpFcn(hObject, eventdata, handles)
%   Terminate ongoing control point move or add operation. Clear
%   coordinate text below plot and update display.

update = strcmpi(handles.updown, 'down');
handles.updown = 'up';      handles.node = 0;
guidata(hObject, handles);
if update
   set(handles.input_text, 'String', '');
   set(handles.output_text, 'String', '');
   render(handles);
end
```

一般情况下，图形回调会在响应一个图形对象或窗口而不是一个活动的uicontrol对象时启动。更明确地，

- 当用户在图形中使用光标而不是在一个活动的uicontrol上（如按钮或弹出菜单）按下鼠标按钮时，执行WindowButtonDownFcn。
- 当用户在图形窗口内移动按下按钮的鼠标时，执行WindowButtonMotionFcn。
- 当用户在图形内而不是在活动的uicontrol上按下鼠标按钮后再释放鼠标按钮时，执行WindowButtonUpFcn。

ice的图形回调的用途和行为已在代码中进行了说明（通过注释）。关于最终的实现，我们做如下的基本观察：

1. 因为ice_WindowButtonDownFcn是在ice图形中(除了在一个活动的图形对象上之外)点击鼠标按钮时调用的,所以回调函数的首要工作是了解光标是否在ice的图形区域内(即curve_axes对象的范围)。若光标在这一区域之外,则应忽略鼠标动作。该测试由内部函数cursor来执行,函数cursor的代码清单已在前一节中给出。在函数cursor中,语句

```
p = get(h, 'CurrentPoint');
```

返回当前光标的坐标。变量自ice_WindowButtonDownFcn传递,并作为输入参量hObject。在所有图形回调中,hObject是图形请求服务的句柄。属性'CurrentPoint'包含有与图形相关光标位置,该位置以两元素行向量[x y]表示。

2. 因为ice是为有两个和三个按钮的鼠标而设计的,所以ice_WindowButtonDownFcn必须确定每一个回调是由哪个鼠标按钮引起的。正如我们在代码中所看到的那样,这是由使用了图形的'SelectionType'属性的一个switch结构来实现的。情形'normal','extent'和'alt'分别对应点击三按钮鼠标的左、中和右按钮(或者点击两按钮鼠标的左、shift加左和control加左按钮),并用于触发加入控制点、移动控制点和删除控制点的操作。

3. 在每次修改控制点时,会更新显示的ICE映射函数(通过内部函数graph),但句柄存储在handles.output中的输出图形仅在鼠标按钮释放后才被更新。这是因为由内部函数render所执行的输出图像的计算是很费时的。这种计算涉及了分别映射输入图像的三个彩色分量,由"全分量"曲线重新映射每一个分量,以及将映射后的分量转换为可供显示的RGB彩色空间。注意,若没有适当的预防措施,在这个冗长的映射处理过程中有可能修改映射函数的控制点。

为了防止出现这种情况,ice控制了其各种回调的中断能力。所有的MATLAB图形对象都有一个中断属性,它决定是否能够中断相应的回调。每一个对象的中断属性默认时为'on',它意味着对象的回调可以被中断。若切换为'off',则在当前不能中断的回调执行期间,任何回调都将忽略(即取消),或者放在事件队列中以待后续处理。中断回调的部署将由将被中断的对象的'BusyAction'属性确定。若'BusyAction'是'cancel',则放弃回调;若是'queue',则在非中断的回调完成之后再处理该回调。

函数ice_WindowButtonUpFcn采用刚刚描述的机制来临时(即在输出图像计算期间)延缓用户处理映射函数控制点的能力。内部函数render中的语句

```
set(handles.ice, 'Interruptible', 'off');
set(handles.ice, 'Pointer', 'watch');


set(handles.ice, 'Pointer', 'arrow');
set(handles.ice, 'Interruptible', 'on');
```

在输出图像的映射和伪彩色及全彩色条期间,将ice图形窗口的'Interruptible'属性设置为'off'。这样就可以在映射被执行期间防止用户修改映射函数控制点。

还要注意的是,图形的'Pointer'属性被设置为'watch',以表示ice较忙,并且在输出计算完成时重置为'arrow'。

B.2.4 对象回调函数

在B.2节开始处的起始GUI M文件的最后9行是对象回调函数的截短形式(stubs)。类似于前一节自动产生的图形回调,它们是初始的无效代码。函数的完整版本如下所示。注意,每一个函数

都用不同的ice uicontrol对象（按钮等）来处理用户交互，并且通过由串'_Callback'来级联其Tag属性进行命名。例如，回调函数负责将显示的映射函数命名为component_popup_Callback。当用户激活（即点击）弹出的选项时，会调用该函数。还要注意的是，输入参量hObject是弹出图形对象的句柄，而不是ice图形的句柄（如前一节的图形回调那样）。ICE的对象回调涉及了最少的代码，并且是自解释的。

```
%-------------------------------------------------------------------%
function component_popup_Callback(hObject, eventdata, handles)
%   Accept color component selection, update component specific
%   parameters on GUI, and draw the selected mapping function.

c = get(hObject, 'Value');
handles.cindex = c;
handles.curve = strcat('set', num2str(c));
guidata(hObject, handles);
set(handles.smooth_checkbox, 'Value', handles.smooth(c));
set(handles.slope_checkbox, 'Value', handles.slope(c));
set(handles.pdf_checkbox, 'Value', handles.pdf(c));
set(handles.cdf_checkbox, 'Value', handles.cdf(c));
graph(handles);

%-------------------------------------------------------------------%
function smooth_checkbox_Callback(hObject, eventdata, handles)
%   Accept smoothing parameter for currently selected color
%   component and redraw mapping function.

if get(hObject, 'Value')
   handles.smooth(handles.cindex) = 1;
   nodes = getfield(handles, handles.curve);
   nodes = spreadout(nodes);
   handles = setfield(handles, handles.curve, nodes);
else
   handles.smooth(handles.cindex) = 0;
end
guidata(hObject, handles);
set(handles.ice, 'Pointer', 'watch');
graph(handles);       render(handles);
set(handles.ice, 'Pointer', 'arrow');

%-------------------------------------------------------------------%
function reset_pushbutton_Callback(hObject, eventdata, handles)
%   Init all display parameters for currently selected color
%   component, make map 1:1, and redraw it.

handles = setfield(handles, handles.curve, [0 0; 1 1]);
c = handles.cindex;
handles.smooth(c) = 0;   set(handles.smooth_checkbox, 'Value', 0);
handles.slope(c) = 0;    set(handles.slope_checkbox, 'Value', 0);
handles.pdf(c) = 0;      set(handles.pdf checkbox, 'Value', 0);
handles.cdf(c) = 0;      set(handles.cdf_checkbox, 'Value', 0);
guidata(hObject, handles);
set(handles.ice, 'Pointer', 'watch');
graph(handles);       render(handles);
set(handles.ice, 'Pointer', 'arrow');
```

```
%-------------------------------------------------------------------------%
function slope_checkbox_Callback(hObject, eventdata, handles)
%   Accept slope clamp for currently selected color component and
%   draw function if smoothing is on.

if get(hObject, 'Value')
   handles.slope(handles.cindex) = 1;
else
   handles.slope(handles.cindex) = 0;
end
guidata(hObject, handles);
if handles.smooth(handles.cindex)
   set(handles.ice, 'Pointer', 'watch');
   graph(handles);      render(handles);
   set(handles.ice, 'Pointer', 'arrow');
end

%-------------------------------------------------------------------------%
function resetall_pushbutton_Callback(hObject, eventdata, handles)
%   Init display parameters for color components, make all maps 1:1,
%   and redraw display.

for c = 1:4
   handles.smooth(c) = 0;        handles.slope(c) = 0;
   handles.pdf(c) = 0;           handles.cdf(c) = 0;
   handles = setfield(handles, ['set' num2str(c)], [0 0; 1 1]);
end
set(handles.smooth_checkbox, 'Value', 0);
set(handles.slope_cheokbox, 'Value', 0);
set(handles.pdf_checkbox, 'Value', 0);
set(handles.cdf_checkbox, 'Value', 0);
guidata(hObject, handles);
set(handles.ice, 'Pointer', 'watch');
graph(handles);      render(handles);
set(handles.ice, 'Pointer', 'arrow');

%-------------------------------------------------------------------------%
function pdf_checkbox_Callback(hObject, eventdata, handles)
%   Accept PDF (probability density function or histogram) display
%   parameter for currently selected color component and redraw
%   mapping function if smoothing is on. If set, clear CDF display.

if get(hObject, 'Value')
   handles.pdf(handles.cindex) = 1;
   set(handles.cdf_checkbox, 'Value', 0);
   handles.cdf(handles.cindex) = 0;
else
   handles.pdf(handles.cindex) = 0;
end
guidata(hObject, handles);     graph(handles);

%-------------------------------------------------------------------------%
function cdf_checkbox_Callback(hObject, eventdata, handles)
%   Accept CDF (cumulative distribution function) display parameter
%   for selected color component and redraw mapping function if
```

```
%   smoothing is on. If set, clear CDF display.

if get(hObject, 'Value')
   handles.cdf(handles.cindex) = 1;
   set(handles.pdf_checkbox, 'Value', 0);
   handles.pdf(handles.cindex) = 0;
else
   handles.cdf(handles.cindex) = 0;
end
guidata(hObject, handles);     graph(handles);

%-------------------------------------------------------------------------%
function mapbar_checkbox_Callback(hObject, eventdata, handles)
%   Accept changes to bar map enable state and redraw bars.

handles.barmap = get(hObject, 'Value');
guidata(hObject, handles);     render(handles);

%-------------------------------------------------------------------------%
function mapimage_checkbox_Callback(hObject, eventdata, handles)
%   Accept changes to the image map state and redraw image.

handles.imagemap = get(hObject, 'Value');
guidata(hObject, handles);     render(handles);
```

附录C　M函数

前言

本附录包含了本书前面未列出的所有M函数的程序清单。函数按字母顺序组织。为了便于寻找函数并大致了解其作用，我们用粗体字显示了每个函数的头两行。

A

```
function f = adpmedian(g, Smax)
%ADPMEDIAN Perform adaptive median filtering.
%   F = ADPMEDIAN(G, SMAX) performs adaptive median filtering of
%   image G.  The median filter starts at size 3-by-3 and iterates up
%   to size SMAX-by-SMAX.  SMAX must be an odd integer greater than 1.

% SMAX must be an odd, positive integer greater than 1.
if (Smax <= 1) | (Smax/2 == round(Smax/2)) | (Smax ~= round(Smax))
   error('SMAX must be an odd integer > 1.')
end
[M, N] = size(g);

% Initial setup.
f = g;
f(:) = 0;
alreadyProcessed = false(size(g));

% Begin filtering.
for k = 3:2:Smax
   zmin = ordfilt2(g, 1, ones(k, k), 'symmetric');
   zmax = ordfilt2(g, k * k, ones(k, k), 'symmetric');
   zmed = medfilt2(g, [k k], 'symmetric');
   processUsingLevelB = (zmed > zmin) & (zmax > zmed) & ...
      ~alreadyProcessed;
   zB = (g > zmin) & (zmax > g);
   outputZxy  = processUsingLevelB & zB;
   outputZmed = processUsingLevelB & ~zB;
   f(outputZxy) = g(outputZxy);
   f(outputZmed) = zmed(outputZmed);

   alreadyProcessed = alreadyProcessed | processUsingLevelB;
   if all(alreadyProcessed(:))
      break;
   end
end

% Output zmed for any remaining unprocessed pixels.  Note that this
% zmed was computed using a window of size Smax-by-Smax, which is
% the final value of k in the loop.
f(~alreadyProcessed) = zmed(~alreadyProcessed);
```

B

```
function rc_new = bound2eight(rc)
%BOUND2EIGHT Convert 4-connected boundary to 8-connected boundary.
%   RC_NEW = BOUND2EIGHT(RC) converts a four-connected boundary to an
%   eight-connected boundary.  RC is a P-by-2 matrix, each row of
%   which contains the row and column coordinates of a boundary
%   pixel.  RC must be a closed boundary; in other words, the last
%   row of RC must equal the first row of RC.  BOUND2EIGHT removes
%   boundary pixels that are necessary for four-connectedness but not
%   necessary for eight-connectedness.  RC_NEW is a Q-by-2 matrix,
%   where Q <= P.

if ~isempty(rc) & ~isequal(rc(1, :), rc(end, :))
   error('Expected input boundary to be closed.');
end

if size(rc, 1) <= 3
   % Degenerate case.
   rc_new = rc;
   return;
end

% Remove last row, which equals the first row.
rc_new = rc(1:end - 1, :);

% Remove the middle pixel in four-connected right-angle turns.  We
% can do this in a vectorized fashion, but we can't do it all at
% once. Similar to the way the 'thin' algorithm works in bwmorph,
% we'll remove first the middle pixels in four-connected turns where
% the row and column are both even; then the middle pixels in all
% the remaining four-connected turns where the row is even and the
% column is odd; then again where the row is odd and the column is
% even; and finally where both the row and column are odd.
remove_locations = compute_remove_locations(rc_new);
field1 = remove_locations & (rem(rc_new(:, 1), 2) == 0) & ...
         (rem(rc_new(:, 2), 2) == 0);
rc_new(field1, :) = [];

remove_locations = compute_remove_locations(rc_new);
field2 = remove_locations & (rem(rc_new(:, 1), 2) == 0) & ...
         (rem(rc_new(:, 2), 2) == 1);
rc_new(field2, :) = [];

remove_locations = compute_remove_locations(rc_new);
field3 = remove_locations & (rem(rc_new(:, 1), 2) == 1) & ...
         (rem(rc_new(:, 2), 2) == 0);
rc_new(field3, :) = [];

remove_locations = compute_remove_locations(rc_new);
field4 = remove_locations & (rem(rc_new(:, 1), 2) == 1) & ...
         (rem(rc_new(:, 2), 2) == 1);
rc_new(field4, :) = [];

% Make the output boundary closed again.
rc_new = [rc_new; rc_new(1, :)];
```

```
%----------------------------------------------------------------------%
function remove = compute_remove_locations(rc)
% Circular diff.
d = [rc(2:end, :); rc(1, :)] - rc;

% Dot product of each row of d with the subsequent row of d,
% performed in circular fashion.
d1 = [d(2:end, :); d(1, :)];
dotprod = sum(d .* d1, 2);

% Locations of N, S, E, and W transitions followed by
% a right-angle turn.
remove = ~all(d, 2) & (dotprod == 0);

% But we really want to remove the middle pixel of the turn.
remove = [remove(end, :); remove(1:end - 1, :)];

if ~any(remove)
   done = 1;
else
   idx = find(remove);
   rc(idx(1), :) = [];
end
```

```
function rc_new = bound2four(rc)
%BOUND2FOUR Convert 8-connected boundary to 4-connected boundary.
%   RC_NEW = BOUND2FOUR(RC) converts an eight-connected boundary to a
%   four-connected boundary.  RC is a P-by-2 matrix, each row of
%   which contains the row and column coordinates of a boundary
%   pixel.  BOUND2FOUR inserts new boundary pixels wherever there is
%   a diagonal connection.

if size(rc, 1) > 1
   % Phase 1: remove diagonal turns, one at a time until they are all gone.
   done = 0;
   rc1 = [rc(end - 1, :); rc];
   while ~done
      d = diff(rc1, 1);
      diagonal_locations = all(d, 2);
      double_diagonals = diagonal_locations(1:end - 1) & ...
           (diff(diagonal_locations, 1) == 0);
      double_diagonal_idx = find(double_diagonals);
      turns = any(d(double_diagonal_idx, :) ~= ...
                  d(double_diagonal_idx + 1, :), 2);
      turns_idx = double_diagonal_idx(turns);
      if isempty(turns_idx)
         done = 1;
      else
         first_turn = turns_idx(1);
         rc1(first_turn + 1, :) = (rc1(first_turn, :) + ...
                                  rc1(first_turn + 2, :)) / 2;
         if first_turn == 1
            rc1(end, :) = rc1(2, :);
         end
```

```
      end
   end
   rc1 = rc1(2:end, :);
end

% Phase 2: insert extra pixels where there are diagonal connections.

rowdiff = diff(rc1(:, 1));
coldiff = diff(rc1(:, 2));

diagonal_locations = rowdiff & coldiff;
num_old_pixels = size(rc1, 1);
num_new_pixels = num_old_pixels + sum(diagonal_locations);
rc_new = zeros(num_new_pixels, 2);

% Insert the original values into the proper locations in the new RC
% matrix.
idx = (1:num_old_pixels)' + [0; cumsum(diagonal_locations)];
rc_new(idx, :) = rc1;

% Compute the new pixels to be inserted.
new_pixel_offsets = [0 1; -1 0; 1 0; 0 -1];
offset_codes = 2 * (1 - (coldiff(diagonal_locations) + 1)/2) + ...
   (2 - (rowdiff(diagonal_locations) + 1)/2);
new_pixels = rc1(diagonal_locations, :) + ...
   new_pixel_offsets(offset_codes, :);

% Where do the new pixels go?
insertion_locations = zeros(num_new_pixels, 1);
insertion_locations(idx) = 1;
insertion_locations = ~insertion_locations;

% Insert the new pixels.
rc_new(insertion_locations, :) = new_pixels;
```

```
function B = bound2im(b, M, N, x0, y0)
%BOUND2IM Converts a boundary to an image.
%   B = BOUND2IM(b) converts b, an np-by-2 or 2-by-np array
%   representing the integer coordinates of a boundary, into a binary
%   image with 1s in the locations defined by the coordinates in b
%   and 0s elsewhere.
%
%   B = BOUND2IM(b, M, N) places the boundary approximately centered
%   in an M-by-N image. If any part of the boundary is outside the
%   M-by-N rectangle, an error is issued.
%
%   B = BOUND2IM(b, M, N, X0, Y0) places the boundary in an image of
%   size M-by-N, with the topmost boundary point located at X0 and
%   the leftmost point located at Y0. If the shifted boundary is
%   outside the M-by-N rectangle, an error is issued. XO and X0 must
%   be positive integers.

[np, nc] = size(b);
if np < nc
   b = b'; % To convert to size np-by-2.
   [np, nc] = size(b);
end
```

```
else
   next_direction_lut = [2 3 4 1];
end

% Values used for marking the starting and boundary pixels.
START    = -1;
BOUNDARY = -2;

% Initialize scratch space in which to record the boundary pixels as
% well as follow the boundary.
scratch = zeros(100, 1);

% Find candidate starting locations for boundaries.
[rr, cc] = find((Lp(2:end-1, :) > 0) & (Lp(1:end-2, :) == 0));
rr = rr + 1;

for k = 1:length(rr)
   r = rr(k);
   c = cc(k);
   if (Lp(r,c) > 0) & (Lp(r - 1, c) == 0) & isempty(B{Lp(r, c)})
      % We've found the start of the next boundary.  Compute its
      % linear offset, record which boundary it is, mark it, and
      % initialize the counter for the number of boundary pixels.
      idx = (c-1)*size(Lp, 1) + r;
      which = Lp(idx);

      scratch(1) = idx;
      Lp(idx) = START;
      numPixels = 1;
      currentPixel = idx;
      initial_departure_direction = [];
      done = 0;
      next_search_direction = 2;
      while ~done
         % Find the next boundary pixel.
         direction = next_search_direction;
         found_next_pixel = 0;
         for k = 1:length(offsets)
            neighbor = currentPixel + offsets(direction);
            if Lp(neighbor) ~= 0
               % Found the next boundary pixel.

               if (Lp(currentPixel) == START) & ...
                       isempty(initial_departure_direction)
                  % We are making the initial departure from
                  % the starting pixel.
                  initial_departure_direction = direction;

               elseif (Lp(currentPixel) == START) & ...
                       (initial_departure_direction == direction)
                  % We are about to retrace our path.
                  % That means we're done.
                  done = 1;
                  found_next_pixel = 1;
                  break;
               end
```

```
                % Take the next step along the boundary.
                next_search_direction = ...
                    next_search_direction_lut(direction);
                found_next_pixel = 1;
                numPixels = numPixels + 1;
                if numPixels > size(scratch, 1)
                   % Double the scratch space.
                   scratch(2*size(scratch, 1)) = 0;
                end
                scratch(numPixels) = neighbor;

                if Lp(neighbor) ~= START
                   Lp(neighbor) = BOUNDARY;
                end

                currentPixel = neighbor;
                break;
             end

             direction = next_direction_lut(direction);
          end

          if ~found_next_pixel
             % If there is no next neighbor, the object must just
             % have a single pixel.
             numPixels = 2;
             scratch(2) = scratch(1);
             done = 1;
          end
       end

       % Convert linear indices to row-column coordinates and save
       % in the output cell array.
       [row, col] = ind2sub(size(Lp), scratch(1:numPixels));
       B{which} = [row - 1, col - 1];
   end
end

if strcmp(dir, 'ccw')
   for k = 1:length(B)
      B{k} = B{k}(end:-1:1, :);
   end
end
```

```
function [s, su] = bsubsamp(b, gridsep)
%BSUBSAMP Subsample a boundary.
%   [S, SU] = BSUBSAMP(B, GRIDSEP) subsamples the boundary B by
%   assigning each of its points to the grid node to which it is
%   closest.  The grid is specified by GRIDSEP, which is the
%   separation in pixels between the grid lines. For example, if
%   GRIDSEP = 2, there are two pixels in between grid lines. So, for
%   instance, the grid points in the first row would be at (1,1),
%   (1,4), (1,6), ..., and similarly in the y direction. The value
%   of GRIDSEP must be an even integer. The boundary is specified by
%   a set of coordinates in the form of an np-by-2 array.  It is
%   assumed that the boundary is one pixel thick.
```

```
%
%   Output S is the subsampled boundary. Output SU is normalized so
%   that the grid separation is unity.  This is useful for obtaining
%   the Freeman chain code of the subsampled boundary.

% Check input.
[np, nc] = size(b);
if np < nc
   error('B must be of size np-by-2.');
end
if gridsep/2 ~= round(gridsep/2)
   error('GRIDSEP must be an even integer.')
end

% Some boundary tracing programs, such as boundaries.m, end with
% the beginning, resulting in a sequence in which the coordinates
% of the first and last points are the same. If this is the case
% in b, eliminate the last point.
if isequal(b(1, :), b(np, :))
   np = np - 1;
   b = b(1:np, :);
end

% Find the max x and y spanned by the boundary.
xmax = max(b(:, 1));
ymax = max(b(:, 2));
% Determine the number of grid lines with gridsep points in
% between them that can fit in the intervals [1,xmax], [1,ymax],
% without any points in b being left over. If points are left
% over, add zeros to extend xmax and ymax so that an integral
% number of grid lines are obtained.
% Size needed in the x-direction:
L = gridsep + 1;
n = ceil(xmax/L);
T = (n - 1)*L + 1;

% Zx is the number of zeros that would be needed to have grid
% lines without any points in b being left over.
Zx = abs(xmax - T - L);
if Zx == L
   Zx = 0;
end
% Number of grid lines in the x-direction, with L pixel spaces
% in between each grid line.
GLx = (xmax + Zx - 1)/L + 1;

% And for the y-direction:
n = ceil(ymax/L);
T = (n - 1)*L + 1;
Zy = abs(ymax - T - L);
if Zy == L
   Zy = 0;
end
GLy = (ymax + Zy - 1)/L + 1;
```

```
% Form vectors of x and y grid locations.
I = 1:GLx;
% Vector of grid line locations intersecting x-axis.
X(I) = gridsep*I + (I - gridsep);
J = 1:GLy;
% Vector of grid line locations intersecting y-axis.
Y(J) = gridsep*J + (J - gridsep);
% Compute both components of the cityblock distance between each
% element of b and all the grid-line intersections.  Assign each
% point to the grid location for which each comp of the cityblock
% distance was less than gridsep/2. Because gridsep is an even
% integer, these assignments are unique. Note the use of meshgrid to
% optimize the code.
DIST = gridsep/2;
[XG, YG] = meshgrid(X, Y);
Q = 1;
for k=1:np
   [I,J] = find(abs(XG - b(k, 1)) <= DIST & abs(YG - b(k, 2)) <= ...
                DIST);
   IL = length(I);
   ord = k*ones(IL, 1); % To keep track of order of input coordinates
   K = Q + IL - 1;
   d1(Q:K, :) = cat(2, X(I), ord);
   d2(Q:K, :) = cat(2, Y(J), ord);
   Q = K + 1;
end
% d is the set of points assigned to the new grid with line
% separation of gridsep. Note that it is formed as d=(d2,d1) to
% compensate for the coordinate transposition inherent in using
% meshgrid (see Chapter 2).
d = cat(2, d2(:, 1), d1); % The second column of d1 is ord.
% Sort the points using the values in ord, which is the last col in
% d.
d = fliplr(d); % So the last column becomes first.
d = sortrows(d);
d = fliplr(d); % Flip back.
% Eliminate duplicate rows in the first two components of
% d to create the output. The cw or ccw order MUST be preserved.
s = d(:, 1:2);
[s, m, n] = unique(s, 'rows');
% Function unique sorts the data--Restore to original order
% by using the contents of m.
s = [s, m];
s = fliplr(s);
s = sortrows(s);
s = fliplr(s);
s = s(:, 1:2);
% Scale to unit grid so that can use directly to obtain Freeman
```

```
f = double(f);
% Initialize I as a column vector.  It will be reshaped later
% into an image.
I = zeros(M*N, 1);
T = varargin{3};
m = varargin{4};
m = m(:)'; % Make sure that m is a row vector.

if length(varargin) == 4
   method = 'euclidean';
elseif length(varargin) == 5
   method = 'mahalanobis';
else
   error('Wrong number of inputs.');
end
switch method
case 'euclidean'
   % Compute the Euclidean distance between all rows of X and m. See
   % Section 12.2 of DIPUM for an explanation of the following
   % expression. D(i) is the Euclidean distance between vector X(i,:)
   % and vector m.
   p = length(f);
   D = sqrt(sum(abs(f - repmat(m, p, 1)).^2, 2));
case 'mahalanobis'
   C = varargin{5};
   D = mahalanobis(f, C, m);
otherwise
   error('Unknown segmentation method.')
end

% D is a vector of size MN-by-1 containing the distance computations
% from all the color pixels to vector m. Find the distances <= T.
J = find(D <= T);

% Set the values of I(J) to 1.  These are the segmented
% color pixels.
I(J) = 1;

% Reshape I into an M-by-N image.
I = reshape(I, M, N);
```

```
function c = connectpoly(x, y)
%CONNECTPOLY Connects vertices of a polygon.
%   C = CONNECTPOLY(X, Y) connects the points with coordinates given
%   in X and Y with straight lines. These points are assumed to be a
%   sequence of polygon vertices organized in the clockwise or
%   counterclockwise direction. The output, C, is the set of points
%   along the boundary of the polygon in the form of an nr-by-2
%   coordinate sequence in the same direction as the input. The last
%   point in the sequence is equal to the first.

v = [x(:), y(:)];

% Close polygon.
if ~isequal(v(end, :), v(1, :))
   v(end + 1, :) = v(1, :);
```

```
end
% Connect vertices.
segments = cell(1, length(v) - 1);
for I = 2:length(v)
   [x, y] = intline(v(I - 1, 1), v(I, 1), v(I - 1, 2), v(I, 2));
   segments{I - 1} = [x, y];
end
c = cat(1, segments{:});
```

D

```
function s = diameter(L)
%DIAMETER Measure diameter and related properties of image regions.
%   S = DIAMETER(L) computes the diameter, the major axis endpoints,
%   the minor axis endpoints, and the basic rectangle of each labeled
%   region in the label matrix L. Positive integer elements of L
%   correspond to different regions. For example, the set of elements
%   of L equal to 1 corresponds to region 1; the set of elements of L
%   equal to 2 corresponds to region 2; and so on.  S is a structure
%   array of length max(L(:)). The fields of the structure array
%   include:
%
%     Diameter
%     MajorAxis
%     MinorAxis
%     BasicRectangle
%
%   The Diameter field, a scalar, is the maximum distance between any
%   two pixels in the corresponding region.
%
%   The MajorAxis field is a 2-by-2 matrix.  The rows contain the row
%   and column coordinates for the endpoints of the major axis of the
%   corresponding region.
%
%   The MinorAxis field is a 2-by-2 matrix.  The rows contain the row
%   and column coordinates for the endpoints of the minor axis of the
%   corresponding region.
%
%   The BasicRectangle field is a 4-by-2 matrix.  Each row contains
%   the row and column coordinates of a corner of the
%   region-enclosing rectangle defined by the major and minor axes.
%
%   For more information about these measurements, see Section 11.2.1
%   of Digital Image Processing, by Gonzalez and Woods, 2nd edition,
%   Prentice Hall.
s = regionprops(L, {'Image', 'BoundingBox'});
for k = 1:length(s)
    [s(k).Diameter, s(k).MajorAxis, perim_r, perim_c] = ...
        compute_diameter(s(k));
    [s(k).BasicRectangle, s(k).MinorAxis] = ...
        compute_basic_rectangle(s(k), perim_r, perim_c);
```

```
end
%-----------------------------------------------------------------------%
function [d, majoraxis, r, c] = compute_diameter(s)
%   [D, MAJORAXIS, R, C] = COMPUTE_DIAMETER(S) computes the diameter
%   and major axis for the region represented by the structure S. S
%   must contain the fields Image and BoundingBox.  COMPUTE_DIAMETER
%   also returns the row and column coordinates (R and C) of the
%   perimeter pixels of s.Image.

% Compute row and column coordinates of perimeter pixels.
[r, c] = find(bwperim(s.Image));
r = r(:);
c = c(:);
[rp, cp] = prune_pixel_list(r, c);

num_pixels = length(rp);
switch num_pixels
case 0
   d = -Inf;
   majoraxis = ones(2, 2);

case 1
   d = 0;
   majoraxis = [rp cp; rp cp];

case 2
   d = (rp(2) - rp(1))^2 + (cp(2) - cp(1))^2;
   majoraxis = [rp cp];

otherwise
   % Generate all combinations of 1:num_pixels taken two at at time.
   % Method suggested by Peter Acklam.
   [idx(:, 2) idx(:, 1)] = find(tril(ones(num_pixels), -1));
   rr = rp(idx);
   cc = cp(idx);

   dist_squared = (rr(:, 1) - rr(:, 2)).^2 + ...
      (cc(:, 1) - cc(:, 2)).^2;
   [max_dist_squared, idx] = max(dist_squared);
   majoraxis = [rr(idx,:)' cc(idx,:)'];

   d = sqrt(max_dist_squared);

   upper_image_row = s.BoundingBox(2) + 0.5;
   left_image_col = s.BoundingBox(1) + 0.5;

   majoraxis(:, 1) = majoraxis(:, 1) + upper_image_row - 1;
   majoraxis(:, 2) = majoraxis(:, 2) + left_image_col - 1;
end
%-----------------------------------------------------------------------%
function [basicrect, minoraxis] = compute_basic_rectangle(s, ...
                                                perim_r, perim_c)
%   [BASICRECT,MINORAXIS] = COMPUTE_BASIC_RECTANGLE(S, PERIM_R,
%   PERIM_C) computes the basic rectangle and the minor axis
%   end-points for the region represented by the structure S.  S must
%   contain the fields Image, BoundingBox, MajorAxis, and
```

```
left = min(c);
right = max(c);
% Which points are inside the upper circle?
x = (left + right)/2;
y = top;
radius = bottom - top;
inside_upper = ( (c - x).^2 + (r - y).^2 ) < radius^2;
% Which points are inside the lower circle?
y = bottom;
inside_lower = ( (c - x).^2 + (r - y).^2 ) < radius^2;
% Which points are inside the left circle?
x = left;
y = (top + bottom)/2;
radius = right - left;
inside_left = ( (c - x).^2 + (r - y).^2 ) < radius^2;
% Which points are inside the right circle?
x = right;
inside_right = ( (c - x).^2 + (r - y).^2 ) < radius^2;
% Eliminate points that are inside all four circles.
delete_idx = find(inside_left & inside_right & ...
                  inside_upper & inside_lower);
r(delete_idx) = [];
c(delete_idx) = [];
```

F

```
function  c = fchcode(b, conn, dir)
%FCHCODE Computes the Freeman chain code of a boundary.
%   C = FCHCODE(B) computes the 8-connected Freeman chain code of a
%   set of 2-D coordinate pairs contained in B, an np-by-2 array. C
%   is a structure with the following fields:
%
%      c.fcc     = Freeman chain code (1-by-np)
%      c.diff    = First difference of code c.fcc (1-by-np)
%      c.mm      = Integer of minimum magnitude from c.fcc (1-by-np)
%      c.diffmm  = First difference of code c.mm (1-by-np)
%      c.x0y0    = Coordinates where the code starts (1-by-2)
%
%   C = FCHCODE(B, CONN) produces the same outputs as above, but
%   with the code connectivity specified in CONN. CONN can be 8 for
%   an 8-connected ohain code, or CONN can be 4 for a 4-connected
%   chain code. Specifying CONN=4 is valid only if the input
%   sequence, B, contains transitions with values 0, 2, 4, and 6,
%   exclusively.
%
%   C = FHCODE(B, CONN, DIR) produces the same outputs as above, but,
%   in addition, the desired code direction is specified. Values for
%   DIR can be:
%
%      'same'     Same as the order of the sequence of points in b.
```

```
%                  This is the default.
%
%      'reverse'  Outputs the code in the direction opposite to the
%                  direction of the points in B.  The starting point
%                  for each DIR is the same.
%
%   The elements of B are assumed to correspond to a 1-pixel-thick,
%   fully-connected, closed boundary. B cannot contain duplicate
%   coordinate pairs, except in the first and last positions, which
%   is a common feature of boundary tracing programs.
%
%   FREEMAN CHAIN CODE REPRESENTATION
%   The table on the left shows the 8-connected Freeman chain codes
%   corresponding to allowed deltax, deltay pairs. An 8-chain is
%   converted to a 4-chain if (1) if conn = 4; and (2) only
%   transitions 0, 2, 4, and 6 occur in the 8-code.  Note that
%   dividing 0, 2, 4, and 6 by 2 produce the 4-code.
%
%       -----------------------  --------------
%       deltax | deltay | 8-code  corresp 4-code
%       -----------------------  --------------
%         0        1       0            0
%        -1        1       1
%        -1        0       2            1
%        -1       -1       3
%         0       -1       4            2
%         1       -1       5
%         1        0       6            3
%         1        1       7
%       -----------------------  --------------
%
%   The formula z = 4*(deltax + 2) + (deltay + 2) gives the following
%   sequence corresponding to rows 1-8 in the preceding table: z =
%   11,7,6,5,9,13,14,15. These values can be used as indices into the
%   table, improving the speed of computing the chain code. The
%   preceding formula is not unique, but it is based on the smallest
%   integers (4 and 2) that are powers of 2.

% Preliminaries.
if nargin == 1
   dir = 'same';
   conn = 8;
elseif nargin == 2
   dir = 'same';
elseif nargin == 3
   % Nothing to do here.
else
   error('Incorrect number of inputs.')
end
[np, nc] = size(b);
if np < nc
```

```
   error('B must be of size np-by-2.');
end
% Some boundary tracing programs, such as boundaries.m, output a
% sequence in which the coordinates of the first and last points are
% the same. If this is the case, eliminate the last point.
if isequal(b(1, :), b(np, :))
   np = np - 1;
   b = b(1:np, :);
end
% Build the code table using the single indices from the formula
% for z given above:
C(11)=0; C(7)=1; C(6)=2; C(5)=3; C(9)=4;
C(13)=5; C(14)=6; C(15)=7;
% End of Preliminaries.
% Begin processing.
x0 = b(1, 1);
y0 = b(1, 2);
c.x0y0 = [x0, y0];
% Make sure the coordinates are organized sequentially:
% Get the deltax and deltay between successive points in b. The
% last row of a is the first row of b.
a = circshift(b, [-1, 0]);
% DEL = a - b is an nr-by-2 matrix in which the rows contain the
% deltax and deltay between successive points in b. The two
% components in the kth row of matrix DEL are deltax and deltay
% between point (xk, yk) and (xk+1, yk+1).  The last row of DEL
% contains the deltax and deltay between (xnr, ynr) and (x1, y1),
% (i.e., between the last and first points in b).
DEL = a - b;
% If the abs value of either (or both) components of a pair
% (deltax, deltay) is greater than 1, then by definition the curve
% is broken (or the points are out of order), and the program
% terminates.
if any(abs(DEL(:, 1)) > 1) | any(abs(DEL(:, 2)) > 1);
   error('The input curve is broken or points are out of order.')
end
% Create a single index vector using the formula described above.
z = 4*(DEL(:, 1) + 2) + (DEL(:, 2) + 2);
% Use the index to map into the table. The following are
% the Freeman 8-chain codes, organized in a 1-by-np array.
fcc = C(z);
% Check if direction of code sequence needs to be reversed.
if strcmp(dir, 'reverse')
   fcc = coderev(fcc); % See below for function coderev.
end
% If 4-connectivity is specified, check that all components
% of fcc are 0, 2, 4, or 6.
```

```
if conn == 4
   val = find(fcc == 1 | fcc == 3 | fcc == 5 | fcc ==7 );
   if isempty(val)
      fcc = fcc./2;
   else
      warning('The specified 4-connected code cannot be satisfied.')
   end
end

% Freeman chain code for structure output.
c.fcc = fcc;

% Obtain the first difference of fcc.
c.diff = codediff(fcc,conn); % See below for function codediff.

% Obtain code of the integer of minimum magnitude.
c.mm = minmag(fcc); % See below for function minmag.

% Obtain the first difference of fcc
c.diffmm = codediff(c.mm, conn);

%-------------------------------------------------------------------%
function cr = coderev(fcc)
%   Traverses the sequence of 8-connected Freeman chain code fcc in
%   the opposite direction, changing the values of each code
%   segment. The starting point is not changed. fcc is a 1-by-np
%   array.

% Flip the array left to right.  This redefines the starting point
% as the last point and reverses the order of "travel" through the
% code.
cr = fliplr(fcc);

% Next, obtain the new code values by traversing the code in the
% opposite direction. (0 becomes 4, 1 becomes 5, ... , 5 becomes 1,
% 6 becomes 2, and 7 becomes 3).
ind1 = find(0 <= cr & cr <= 3);
ind2 = find(4 <= cr & cr <= 7);
cr(ind1) = cr(ind1) + 4;
cr(ind2) = cr(ind2) - 4;

%-------------------------------------------------------------------%
function z = minmag(c)
%MINMAG Finds the integer of minimum magnitude in a chain code.
%   Z = MINMAG(C) finds the integer of minimum magnitude in a given
%   4- or 8-connected Freeman chain code, C. The code is assumed to
%   be a 1-by-np array.

% The integer of minimum magnitude starts with min(c), but there
% may be more than one such value. Find them all,
I = find(c == min(c));
% and shift each one left so that it starts with min(c).
J = 0;
A = zeros(length(I), length(c));
for k = I;
   J = J + 1;
```

```
   A(J, :) = circshift(c,[0 -(k-1)]);
end

% Matrix A contains all the possible candidates for the integer of
% minimum magnitude. Starting with the 2nd column, succesively find
% the minima in each column of A. The number of candidates decreases
% as the seach moves to the right on A.  This is reflected in the
% elements of J.  When length(J)=1, one candidate remains.  This is
% the integer of minimum magnitude.
[M, N] = size(A);
J = (1:M)';
for k = 2:N
   D(1:M, 1) = Inf;
   D(J, 1) = A(J, k);
   amin = min(A(J, k));
   J = find(D(:, 1) == amin);
   if length(J)==1
      z = A(J, :);
      return
   end
end
%-------------------------------------------------------------------%
function d = codediff(fcc, conn)
%CODEDIFF Computes the first difference of a chain code.
%   D = CODEDIFF(FCC) computes the first difference of code, FCC. The
%   code FCC is treated as a circular sequence, so the last element
%   of D is the difference between the last and first elements of
%   FCC.  The input code is a 1-by-np vector.
%
%   The first difference is found by oounting the number of direction
%   changes (in a counter-clockwise direction) that separate two
%   adjacent elements of the code.

sr = circshift(fcc, [0, -1]); % Shift input left by 1 location.
delta = sr - fcc;
d = delta;
I = find(delta < 0);

type = conn;
switch type
case 4 % Code is 4-connected
   d(I) = d(I) + 4;
case 8 % Code is 8-connected
   d(I) = d(I) + 8;
end
```

G

```
function g = gscale(f, varargin)
%GSCALE Scales the intensity of the input image.
%   G = GSCALE(F, 'full8') scales the intensities of F to the full
%   8-bit intensity range [0, 255].  This is the default if there is
%   only one input argument.
%
```

```
%   is an M-by-N-by-n stack array of n registered images of size
%   M-by-N each (see Fig. 11.24). The extracted vectors are arranged
%   as the rows of array X. Input MASK is an M-by-N logical or
%   numeric image with nonzero values (1s if it is a logical array)
%   in the locations where elements of S are to be used in forming X
%   and 0s in locations to be ignored. The number of row vectors in X
%   is equal to the number of nonzero elements of MASK. If MASK is
%   omitted, all M*N locations are used in forming X.  A simple way to
%   obtain MASK interactively is to use function roipoly. Finally, R
%   is an array whose rows are the 2-D coordinates containing the
%   region locations in MASK from which the vectors in S were
%   extracted to form X.

% Preliminaries.
[M, N, n] = size(S);
if nargin == 1
   MASK = true(M, N);
else
   MASK = MASK ~= 0;
end

% Find the set of locations where the vectors will be kept before
% MASK is changed later in the program.
[I, J] = find(MASK);
R = [I, J];

% Now find X.

% First reshape S into X by turning each set of n values along the third
% dimension of S so that it becomes a row of X. The order is from top to
% bottom along the first column, the second column, and so on.
Q = M*N;
X = reshape(S, Q, n);

% Now reshape MASK so that it corresponds to the right locations
% vertically along the elements of X.
MASK = reshape(MASK, Q, 1);

% Keep the rows of X at locations where MASK is not 0.
X = X(MASK, :);
```

```
function [x, y] = intline(x1, x2, y1, y2)
%INTLINE Integer-coordinate line drawing algorithm.
%   [X, Y] = INTLINE(X1, X2, Y1, Y2) computes an
%   approximation to the line segment joining (X1, Y1) and
%   (X2, Y2) with integer coordinates.  X1, X2, Y1, and Y2
%   should be integers.  INTLINE is reversible; that is,
%   INTLINE(X1, X2, Y1, Y2) produces the same results as
%   FLIPUD(INTLINE(X2, X1, Y2, Y1)).
%   Copyright 1993-2002 The MathWorks, Inc. Used with permission.
%   $Revision: 5.11 $  $Date: 2002/03/15 15:57:47 $

dx = abs(x2 - x1);
dy = abs(y2 - y1);

% Check for degenerate case.
if ((dx == 0) & (dy == 0))
```

```
    x = x1;
    y = y1;
    return;
end

flip = 0;
if (dx >= dy)
    if (x1 > x2)
       % Always "draw" from left to right.
       t = x1; x1 = x2; x2 = t;
       t = y1; y1 = y2; y2 = t;
       flip = 1;
    end
    m = (y2 - y1)/(x2 - x1);
    x = (x1:x2).';
    y = round(y1 + m*(x - x1));
else
    if (y1 > y2)
       % Always "draw" from bottom to top.
       t = x1; x1 = x2; x2 = t;
       t = y1; y1 = y2; y2 = t;
       flip = 1;
    end
    m = (x2 - x1)/(y2 - y1);
    y = (y1:y2).';
    x = round(x1 + m*(y - y1));
end

if (flip)
    x = flipud(x);
    y = flipud(y);
end
```

```
function phi = invmoments(F)
%INVMOMENTS Compute invariant moments of image.
%   PHI = INVMOMENTS(F) computes the moment invariants of the image
%   F. PHI is a seven-element row vector containing the moment
%   invariants as defined in equations (11.3-17) through (11.3-23) of
%   Gonzalez and Woods, Digital Image Processing, 2nd Ed.
%
%   F must be a 2-D, real, nonsparse, numeric or logical matrix.

if (ndims(F) ~= 2) | issparse(F) | ~isreal(F) | ~(isnumeric(F) | ...
                                                   islogical(F))
   error(['F must be a 2-D, real, nonsparse, numeric or logical '...
          'matrix.']);
end

F = double(F);

phi = compute_phi(compute_eta(compute_m(F)));

%-------------------------------------------------------------------%
function m = compute_m(F)

[M, N] = size(F);
```

```
[x, y] = meshgrid(1:N, 1:M);
% Turn x, y, and F into column vectors to make the summations a bit
% easier to compute in the following.
x = x(:);
y = y(:);
F = F(:);
% DIP equation (11.3-12)
m.m00 = sum(F);
% Protect against divide-by-zero warnings.
if (m.m00 == 0)
   m.m00 = eps;
end
% The other central moments:
m.m10 = sum(x .* F);
m.m01 = sum(y .* F);
m.m11 = sum(x .* y .* F);
m.m20 = sum(x.^2 .* F);
m.m02 = sum(y.^2 .* F);
m.m30 = sum(x.^3 .* F);
m.m03 = sum(y.^3 .* F);
m.m12 = sum(x .* y.^2 .* F);
m.m21 = sum(x.^2 .* y .* F);
%-------------------------------------------------------------------%
function e = compute_eta(m)
% DIP equations (11.3-14) through (11.3-16).
xbar = m.m10 / m.m00;
ybar = m.m01 / m.m00;
e.eta11 = (m.m11 - ybar*m.m10) / m.m00^2;
e.eta20 = (m.m20 - xbar*m.m10) / m.m00^2;
e.eta02 = (m.m02 - ybar*m.m01) / m.m00^2;
e.eta30 = (m.m30 - 3 * xbar * m.m20 + 2 * xbar^2 * m.m10) / m.m00^2.5;
e.eta03 = (m.m03 - 3 * ybar * m.m02 + 2 * ybar^2 * m.m01) / m.m00^2.5;
e.eta21 = (m.m21 - 2 * xbar * m.m11 - ybar * m.m20 + ...
           2 * xbar^2 * m.m01) / m.m00^2.5;
e.eta12 = (m.m12 - 2 * ybar * m.m11 - xbar * m.m02 + ...
           2 * ybar^2 * m.m10) / m.m00^2.5;
%-------------------------------------------------------------------%
function phi = compute_phi(e)
% DIP equations (11.3-17) through (11.3-23).
phi(1) = e.eta20 + e.eta02;
phi(2) = (e.eta20 - e.eta02)^2 + 4*e.eta11^2;
phi(3) = (e.eta30 - 3*e.eta12)^2 + (3*e.eta21 - e.eta03)^2;
phi(4) = (e.eta30 + e.eta12)^2 + (e.eta21 + e.eta03)^2;
phi(5) = (e.eta30 - 3*e.eta12) * (e.eta30 + e.eta12) * ...
         ( (e.eta30 + e.eta12)^2 - 3*(e.eta21 + e.eta03)^2 ) + ...
         (3*e.eta21 - e.eta03) * (e.eta21 + e.eta03) * ...
         ( 3*(e.eta30 + e.eta12)^2 - (e.eta21 + e.eta03)^2 );
phi(6) = (e.eta20 - e.eta02) * ( (e.eta30 + e.eta12)^2 - ...
```

```
y = c(I);
% [x', y'] is a length(I)-by-2 array.  Each member of this array is
% the left, top corner of a black cell of size cellsize-by-cellsize.
% Fill the cells with black to form a closed border of black cells
% around interior points. These cells are the cellular complex.
for k = 1:length(I)
   B(x(k):x(k) + cellsize-1, y(k):y(k) + cellsize-1) = 1;
end
BF = imfill(B, 'holes');
% Extract the points interior to the black border. This is the region
% of interest around which the MPP will be found.
B = BF & (~B);
% Extract the 4-connected boundary.
B = boundaries(B, 4, 'cw');
% Find the largest one in case of parasitic regions.
J = cellfun('length', B);
I = find(J == max(J));
B = B{I(1)};
% Function boundaries outputs the last coordinate pair equal to the
% first.  Delete it.
B = B(1:end-1,:);
% Obtain the xy coordinates of the boundary.
x = B(:, 1);
y = B(:, 2);
% Find the smallest x-coordinate and corresponding
% smallest y-coordinate.
cx = find(x == min(x));
cy = find(y == min(y(cx)));
% The cell with top leftmost corner at (x1, y1) below is the first
% point considered by the algorithm.  The remaining points are
% visited in the clockwise direction starting at (x1, y1).
x1 = x(cx(1));
y1 = y(cy(1));
% Scroll data so that the first point is (x1, y1).
I = find(x == x1 & y == y1);
x = circshift(x, [-(I - 1), 0]);
y = circshift(y, [-(I - 1), 0]);
% The same shift applies to B.
B = circshift(B, [-(I - 1), 0]);
% Get the Freeman chain code.  The first row of B is the required
% starting point. The first element of the code is the transition
% between the 1st and 2nd element of B, the second element of
% the code is the transition between the 2nd and 3rd elements of B,
% and so on.  The last element of the code is the transition between
% the last and 1st elements of B. The elements of B form a cw
% sequence (see above), so we use 'same' for the direction in
% function fchcode.
code = fchcode(B, 4, 'same');
```

```
code = code.fcc;
% Follow the code sequence to extract the Black Dots, BD, (convex
% corners) and White Dots, WD, (concave corners). The transitions are
% as follows: 0-to-1=WD; 0-to-3=BD; 1-to-0=BD; 1-to-2=WD; 2-to-1=BD;
% 2-to-3=WD; 3-to-0=WD; 3-to-2=dot.  The formula t = 2*first - second
% gives the following unique values for these transitions: -1, -3, 2,
% 0, 3, 1, 6, 4.  These are applicable to travel in the cw direction.
% The WD's are displaced one-half a diagonal from the BD's to form
% the half-cell expansion required in the algorithm.
% Vertices will be computed as array "vertices" of dimension nv-by-3,
% where nv is the number of vertices. The first two elements of any
% row of array vertices are the (x,y) coordinates of the vertex
% corresponding to that row, and the third element is 1 if the
% vertex is convex (BD) or 2 if it is concave (WD). The first vertex
% is known to be convex, so it is black.
vertices = [x1, y1, 1];
n = 1;
k = 1;
for k = 2:length(code)
   if code(k - 1) ~= code(k)
      n = n + 1;
      t = 2*code(k-1) - code(k); % t = value of formula.
      if t == -3 | t == 2 | t == 3 | t == 4 % Convex: Black Dots.
         vertices(n, 1:3) = [x(k), y(k), 1];
      elseif t == -1 | t == 0 | t == 1 | t == 6 % Concave: White Dots.
         if t == -1
            vertices(n, 1:3) = [x(k) - cellsize, y(k) - cellsize,2];
         elseif t==0
            vertices(n, 1:3) = [x(k) + cellsize, y(k) - cellsize,2];
         elseif t==1
            vertices(n, 1:3) = [x(k) + cellsize, y(k) + cellsize,2];
         else
            vertices(n, 1:3) = [x(k) - cellsize, y(k) + cellsize,2];
         end
      else
         % Nothing to do here.
      end
   end
end
% The rest of minperpoly.m processes the vertices to
% arrive at the MPP.
flag = 1;
while flag
   % Determine which vertices lie on or inside the
   % polygon whose vertices are the Black Dots. Delete all
   % other points.
   I = find(vertices(:, 3) == 1);
   xv = vertices(I, 1); % Coordinates of the Black Dots.
   yv = vertices(I, 2);
```

```
    X = vertices(:, 1); % Coordinates of all vertices.
    Y = vertices(:, 2);
    IN = inpolygon(X, Y, xv, yv);
    I = find(IN ~= 0);
    vertices = vertices(I, :);

    % Now check for any Black Dots that may have been turned into
    % concave vertices after the previous deletion step. Delete
    % any such Black Dots and recompute the polygon as in the
    % previous section of code. When no more changes occur, set
    % flag to 0, which causes the loop to terminate.
    x = vertices(:, 1);
    y = vertices(:, 2);
    angles = polyangles(x, y); % Find all the interior angles.
    I = find(angles > 180 & vertices(:, 3) == 1);
    if isempty(I)
       flag = 0;
    else
       J = 1:length(vertices);
       for k = 1:length(I)
          K = find(J ~= I(k));
          J = J(K);
       end
       vertices = vertices(J, :);
    end
end

% Final pass to delete the vertices with angles of 180 degrees.
x = vertices(:, 1);
y = vertices(:, 2);
angles = polyangles(x, y);
I = find(angles ~= 180);

% Vertices of the MPP:
x = vertices(I, 1);
y = vertices(I, 2);
```

P

```
function B = pixeldup(A, m, n)
%PIXELDUP Duplicates pixels of an image in both directions.
%   B = PIXELDUP(A, M, N) duplicates each pixel of A M times in the
%   vertical direction and N times in the horizontal direction.
%   Parameters M and N must be integers.  If N is not included, it
%   defaults to M.

% Check inputs.
if nargin < 2
   error('At least two inputs are required.');
end
if nargin == 2
   n = m;
end

% Generate a vector with elements 1:size(A, 1).
```

```
u = 1:size(A, 1);

% Duplicate each element of the vector m times.
m = round(m); % Protect against nonintergers.
u = u(ones(1, m), :);
u = u(:);

% Now repeat for the other direction.
v = 1:size(A, 2);
n = round(n);
v = v(ones(1, n), :);
v = v(:);
B = A(u, v);
```

```
function angles = polyangles(x, y)
%POLYANGLES Computes internal polygon angles.
%   ANGLES = POLYANGLES(X, Y) computes the interior angles (in
%   degrees) of an arbitrary polygon whose vertices are given in
%   [X, Y], ordered in a clockwise manner.  The program eliminates
%   duplicate adjacent rows in [X Y], except that the first row may
%   equal the last, so that the polygon is closed.

% Preliminaries.
[x y] = dupgone(x, y); % Eliminate duplicate vertices.
xy = [x(:) y(:)];
if isempty(xy)
   % No vertices!
   angles = zeros(0, 1);
   return;
end

if size(xy, 1) == 1 | ~isequal(xy(1, :), xy(end, :))
   % Close the polygon
   xy(end + 1, :) = xy(1, :);
end
% Precompute some quantities.
d = diff(xy, 1);
v1 = -d(1:end, :);
v2 = [d(2:end, :); d(1, :)];
v1_dot_v2 = sum(v1 .* v2, 2);
mag_v1 = sqrt(sum(v1.^2, 2));
mag_v2 = sqrt(sum(v2.^2, 2));

% Protect against nearly duplicate vertices; output angle will be 90
% degrees for such cases. The "real" further protects against
% possible small imaginary angle components in those cases.
mag_v1(~mag_v1) = eps;
mag_v2(~mag_v2) = eps;
angles = real(acos(v1_dot_v2 ./ mag_v1 ./ mag_v2) * 180 / pi);

% The first angle computed was for the second vertex, and the
% last was for the first vertex. Scroll one position down to
% make the last vertex be the first.
angles = circshift(angles, [1, 0]);

% Now determine if any vertices are concave and adjust the angles
```

```
% accordingly.
sgn = convex_angle_test(xy);

% Any element of sgn that's -1 indicates that the angle is
% conoave. The corresponding angles have to be subtracted
% from 360.
I = find(sgn == -1);
angles(I) = 360 - angles(I);

%-------------------------------------------------------------------%
function sgn = convex_angle_test(xy)
%   The rows of array xy are ordered vertices of a polygon. If the
%   kth angle is convex (>0 and <= 180 degress) then sgn(k) =
%   1. Otherwise sgn(k) = -1. This function assumes that the first
%   vertex in the list is convex, and that no other vertex has a
%   smaller value of x-coordinate. These two conditions are true in
%   the first vertex generated by the MPP algorithm. Also the
%   vertices are assumed to be ordered in a clockwise sequence, and
%   there can be no duplicate vertices.
%
%   The test is based on the fact that every convex vertex is on the
%   positive side of the line passing through the two vertices
%   immediately following each vertex being considered.  If a vertex
%   is concave then it lies on the negative side of the line joining
%   the next two vertices. This property is true also if positive and
%   negative are interchanged in the preceding two sentences.

% It is assumed that the polygon is closed.  If not, close it.
if size(xy, 1) == 1 | ~isequal(xy(1, :), xy(end, :))
   xy(end + 1, :) = xy(1, :);
end

% Sign convention: sgn = 1 for convex vertices (i.e, interior angle > 0
% and <= 180 degrees), sgn = -1 for concave vertices.
% Extreme points to be used in the following loop.  A 1 is appended
% to perform the inner (dot) product with w, which is 1-by-3 (see
% below).
L = 10^25;
top_left = [-L, -L, 1];
top_right = [-L, L, 1];
bottom_left = [L, -L, 1];
bottom_right = [L, L, 1];

sgn = 1; % The first vertex is known to be convex.

% Start following the vertices.
for k = 2:length(xy) - 1
   pfirst= xy(k - 1, :);
   psecond = xy(k, :); % This is the point tested for convexity.
   pthird = xy(k + 1, :);
   % Get the coefficients of the line (polygon edge) passing
   % through pfirst and psecond.
   w = polyedge(pfirst, psecond);

   % Establish the positive side of the line w1x + w2y + w3 = 0.
```

```
    % The positive side of the line should be in the right side of the
    % vector (psecond - pfirst).  deltax and deltay of this vector
    % give the direction of travel. This establishes which of the
    % extreme points (see above) should be on the + side. If that
    % point is on the negative side of the line, then w is replaced by -w.
    deltax = psecond(:, 1) - pfirst(:, 1);
    deltay = psecond(:, 2) - pfirst(:, 2);
    if deltax == 0 & deltay == 0
       error('Data into convexity test is 0 or duplicated.')
    end
    if deltax <= 0  & deltay >= 0 % Bottom_right should be on + side.
       vector_product = dot(w, bottom_right); % Inner product.
       w = sign(vector_product)*w;
    elseif deltax <= 0 & deltay <= 0 % Top_right should be on + side.
       vector_product = dot(w, top_right);
       w = sign(vector_product)*w;
    elseif deltax >= 0 & deltay <= 0  % Top_left should be on + side.
       vector_product = dot(w, top_left);
       w = sign(vector_product)*w;
    else % deltax >= 0 & deltay >= 0, so bottom_left should be on + side.
       vector_product = dot(w, bottom_left);
       w = sign(vector_product)*w;
    end
    % For the vertex at psecond to be convex, pthird has to be on the
    % positive side of the line.
    sgn(k) = 1;
    if (w(1)*pthird(:, 1) + w(2)*pthird(:, 2) + w(3)) < 0
       sgn(k) = -1;
    end
end
%-------------------------------------------------------------------%
function w = polyedge(p1, p2)
%   Outputs the coefficients of the line passing through p1 and
%   p2. The line is of the form w1x + w2y + w3 = 0.
x1 = p1(:, 1);  y1 = p1(:, 2);
x2 = p2(:, 1);  y2 = p2(:, 2);
if x1==x2
   w2 = 0;
   w1 = -1/x1;
   w3 = 1;
elseif y1==y2
   w1 = 0;
   w2 = -1/y1;
   w3 = 1;
elseif x1 == y1 & x2 == y2
   w1 = 1;
   w2 = 1;
   w3 = 0;
else
```

```
%   signature of a given boundary, B, where B is an np-by-2 array
%   (np > 2) containing the (x, y) coordinates of the boundary
%   ordered in a clockwise or counterclockwise direction. The
%   amplitude of the signature as a function of increasing ANGLE is
%   output in ST. (X0,Y0) are the coordinates of the centroid of the
%   boundary. The maximum size of arrays ST and ANGLE is 360-by-1,
%   indicating a maximum resolution of one degree. The input must be
%   a one-pixel-thick boundary obtained, for example, by using the
%   function boundaries. By definition, a boundary is a closed curve.
%
%   [ST, ANGLE, X0, Y0] = SIGNATURE(B) computes the signature, using
%   the centroid as the origin of the signature vector.
%
%   [ST, ANGLE, X0, Y0] = SIGNATURE(B, X0, Y0) computes the boundary
%   using the specified (X0, Y0) as the origin of the signature
%   vector.

% Check dimensions of b.
[np, nc] = size(b);
if (np < nc | nc ~= 2)
   error('B must be of size np-by-2.');
end

% Some boundary tracing programs, such as boundaries.m, end where
% they started, resulting in a sequence in which the coordinates
% of the first and last points are the same. If this is the case,
% in b, eliminate the last point.
if isequal(b(1, :), b(np, :))
   b = b(1:np - 1, :);
   np = np - 1;
end

% Compute parameters.
if nargin == 1
   x0 = round(sum(b(:, 1))/np); % Coordinates of the centroid.
   y0 = round(sum(b(:, 2))/np);
elseif nargin == 3
   x0 = varargin{1};
   y0 = varargin{2};
else
   error('Incorrect number of inputs.');
end

% Shift origin of coord system to (x0, y0)).
b(:, 1) = b(:, 1) - x0;
b(:, 2) = b(:, 2) - y0;

% Convert the coordinates to polar.  But first have to convert the
% given image coordinates, (x, y), to the coordinate system used by
% MATLAB for conversion between Cartesian and polar cordinates.
% Designate these coordinates by (xc, yc). The two coordinate systems
% are related as follows:  xc = y and yc = -x.
xc = b(:, 2);
yc = -b(:, 1);
```

```
[theta, rho] = cart2pol(xc, yc);

% Convert angles to degrees.
theta = theta.*(180/pi);

% Convert to all nonnegative angles.
j = theta == 0; % Store the indices of theta = 0 for use below.
theta = theta.*(0.5*abs(1 + sign(theta)))...
        - 0.5*(-1 + sign(theta)).*(360 + theta);
theta(j) = 0; % To preserve the 0 values.

temp = theta;
% Order temp so that sequence starts with the smallest angle.
% This will be used below in a check for monotonicity.
I = find(temp == min(temp));

% Scroll up so that sequence starts with the smallest angle.
% Use I(1) in case the min is not unique (in this case the
% sequenoe will not be monotonic anyway).
temp = circshift(temp, [-(I(1) - 1), 0]);

% Check for monotonicity, and issue a warning if sequence
% is not monotonic. First determine if sequence is
% cw or ccw.
k1 = abs(temp(1) - temp(2));
k2 = abs(temp(1) - temp(3));
if k2 > k1
   sense = 1; % ccw
elseif k2 < k1
   sense = -1; % cw
else
   warning(['The first 3 points in B do not form a monotonic ' ...
            'sequence.']);
end
% Check the rest of the sequence for monotonicity. Because
% the angles are rounded to the nearest integer later in the
% program, only differences greater than 0.5 degrees are
% considered in the test for monotonicity in the rest of
% the sequence.
flag = 0;
for k = 3:length(temp) - 1
   diff = sense*(temp(k + 1) - temp(k));
   if diff < -.5
      flag = 1;
   end
end
if flag
   warning('Angles do not form a monotonic sequence.');
end

% Round theta to 1 degree increments.
theta = rcund(theta);

% Keep theta and rho together.
tr = [theta, rho];

% Delete duplicate angles.  The unique operation
```

```
% Output the log of the spectrum for easier viewing, scaled to the
% range [0, 1].
S = mat2gray(log(1 + S));
%-------------------------------------------------------------------%
function [xc, yc] = halfcircle(r, x0, y0)
%   Computes the integer coordinates of a half circle of radius r and
%   center at (x0,y0) using one degree increments.
%
%   Goes from 91 to 270 because we want the half circle to be in the
%   region defined by top right and top left quadrants, in the
%   standard image coordinates.

theta=91:270;
theta = theta*pi/180;
[xc, yc] = pol2cart(theta, r);
xc = round(xc)' + x0; % Column vector.
yc = round(yc)' + y0;
%-------------------------------------------------------------------%
function [xr, yr] = radial(x0, y0, x, y);
%   Computes the coordinates of a straight line segment extending
%   from (x0, y0) to (x, y).
%
%   Based on function intline.m. xr and yr are
%   returned as column vectors.
 [xr, yr] = intline(x0, x, y0, y);
```

```
function [v, unv] = statmoments(p,  n)
%STATMOMENTS Computes statistical central moments of image histogram.
%   [W, UNV] = STATMOMENTS(P, N) computes up to the Nth statistical
%   central moment of a histogram whose components are in vector
%   P. The length of P must equal 256 or 65536.
%
%   The program outputs a vector V with V(1) = mean, V(2) = variance,
%   V(3) = 3rd moment, . . . V(N) = Nth central moment. The random
%   variable values are normalized to the range [0, 1], so all
%   moments also are in this range.
%
%   The program also outputs a vector UNV containing the same moments
%   as V, but using un-normalized random variable values (e.g., 0 to
%   255 if length(P) = 2^8). For example, if length(P) = 256 and V(1)
%   = 0.5, then UNV(1) would have the value UNV(1) = 127.5 (half of
%   the [0 255] range).
Lp = length(p);
if (Lp ~= 256) & (Lp ~= 65536)
   error('P must be a 256- or 65536-element vector.');
end
G = Lp - 1;
% Make sure the histogram has unit area, and convert it to a
% column vector.
p = p/sum(p); p = p(:);
```

```
% Form a vector of all the possible values of the
% random variable.
z = 0:G;

% Now normalize the z's to the range [0, 1].
z = z./G;

% The mean.
m = z*p;

% Center random variables about the mean.
z = z - m;

% Compute the central moments.
v = zeros(1, n);
v(1) = m;
for j = 2:n
   v(j) = (z.^j)*p;
end

if nargout > 1
   % Compute the uncentralized moments.
   unv = zeros(1, n);
   unv(1)=m.*G;
   for j = 2:n
      unv(j) = ((z*G).^j)*p;
   end
end
```

```
function [t] = statxture(f, scale)
%STATXTURE Computes statistical measures of texture in an image.
%   T = STATXURE(F, SCALE) computes six measures of texture from an
%   image (region) F. Parameter SCALE is a 6-dim row vector whose
%   elements multiply the 6 corresponding elements of T for scaling
%   purposes. If SCALE is not provided it defaults to all 1s.  The
%   output T is 6-by-1 vector with the following elements:
%      T(1) = Average gray level
%      T(2) = Average contrast
%      T(3) = Measure of smoothness
%      T(4) = Third moment
%      T(5) = Measure of uniformity
%      T(6) = Entropy

if nargin == 1
   scale(1:6) = 1;
else % Make sure it's a row vector.
   scale = scale(:)';
end
% Obtain histogram and normalize it.
p = imhist(f);
p = p./numel(f);
L = length(p);

% Compute the three moments. We need the unnormalized ones
% from function statmoments. These are in vector mu.
[v, mu] = statmoments(p, 3);
```

```
% Compute the six texture measures:
% Average gray level.
t(1) = mu(1);
% Standard deviation.
t(2) = mu(2).^0.5;
% Smoothness.
% First normalize the variance to [0 1] by
% dividing it by (L-1)^2.
varn = mu(2)/(L - 1)^2;
t(3) = 1 - 1/(1 + varn);
% Third moment (normalized by (L - 1)^2 also).
t(4) = mu(3)/(L - 1)^2;
% Uniformity.
t(5) = sum(p.^2);
% Entropy.
t(6) = -sum(p.*(log2(p + eps)));

% Scale the values.
t = t.*scale;
```

X

```
function [B, theta] = x2majoraxis(A, B, type)
%X2MAJORAXIS Aligns coordinate x with the major axis of a region.
%   [B2, THETA] = X2MAJORAXIS(A, B, TYPE) aligns the x-coordinate
%   axis with the major axis of a region or boundary. The y-axis is
%   perpendicular to the x-axis.  The rows of 2-by-2 matrix A are the
%   coordinates of the two end points of the major axis, in the form
%   A = [x1 y1; x2 y2]. On input, B is either a binary image (i.e.,
%   an array of class logical) containing a single region, or it is
%   an np-by-2 set of points representing a (connected) boundary. In
%   the latter case, the first column of B must represent
%   x-coordinates and the second column must represent the
%   corresponding y-coordinates. On output, B contains the same data
%   as the input, but aligned with the major axis. If the input is an
%   image, so is the output; similarly the output is a sequence of
%   coordinates if the input is such a sequence. Parameter THETA is
%   the initial angle between the major axis and the x-axis. The
%   origin of the xy-axis system is at the bottom left; the x-axis is
%   the horizontal axis and the y-axis is the vertical.
%
%   Keep in mind that rotations can introduce round-off errors when
%   the data are converted to integer coordinates, which is a
%   requirement.  Thus, postprocessing (e.g., with bwmorph) of the
%   output may be required to reconnect a boundary.

% Preliminaries.
if islogical(B)
   type = 'region';
elseif size(B, 2) == 2
   type = 'boundary';
   [M, N] = size(B);
   if M < N
```

```
        error('B is boundary. It must be of size np-by-2; np > 2.')
    end
    % Compute centroid for later use. c is a 1-by-2 vector.
    % Its 1st component is the mean of the boundary in the x-direction.
    % The second is the mean in the y-direction.
    c(1) = round((min(B(:, 1)) + max(B(:, 1))/2));
    c(2) = round((min(B(:, 2)) + max(B(:, 2))/2));

    % It is possible for a connected boundary to develop small breaks
    % after rotation. To prevent this, the input boundary is filled,
    % processed as a region, and then the boundary is re-extracted. This
    % guarantees that the output will be a connected boundary.
    m = max(size(B));
    % The following image is of size m-by-m to make sure that there
    % there will be no size truncation after rotation.
    B = bound2im(B,m,m);
    B = imfill(B,'holes');
else
    error('Input must be a boundary or a binary image'.)
end

% Major axis in vector form.
v(1) = A(2, 1) - A(1, 1);
v(2) = A(2, 2) - A(1, 2);
v = v(:);  % v is a col vector

% Unit vector along x-axis.
u = [1; 0];

% Find angle between major axis and x-axis. The angle is
% given by acos of the inner product of u and v divided by
% the product of their norms. Because the inputs are image
% points, they are in the first quadrant.
nv = norm(v);
nu = norm(u);
theta = acos(u'*v/nv*nu);
if theta > pi/2
    theta = -(theta - pi/2);
end
theta = theta*180/pi;  % Convert angle to degrees.

% Rotate by angle theta and crop the rotated image to original size.
B = imrotate(B, theta, 'bilinear', 'crop');

% If the input was a boundary, re-extract it.
if strcmp(type, 'boundary')
    B = boundaries(B);
    B = B{1};
    % Shift so that centroid of the extracted boundary is
    % approx equal to the centroid of the original boundary:
    B(:, 1) = B(:, 1) - min(B(:, 1)) + c(1);
    B(:, 2) = B(:, 2) - min(B(:, 2)) + c(2);
end
```

参考文献

Gonzalez, R. C. and Woods, R. E. [2002]. *Digital Image Processing*, 2nd ed., Prentice Hall, Upper Saddle River, NJ.

Hanselman, D. and Littlefield, B. R. [2001]. *Mastering MATLAB 6*, Prentice Hall, Upper Saddle River, NJ.

Image Processing Toolbox, Users Guide, Version 4. [2003], The MathWorks, Inc., Natick, MA.

Using MATLAB, Version 6.5 [2002], The MathWorks, Inc., Natick, MA.

Other References Cited:

Acklam, P. J. [2002]. "MATLAB Array Manipulation Tips and Tricks." Available for download at http://home.online.no/~pjacklam/matlab/doc/mtt/ and also from www.prenhall.com/gonzalezwoodseddins.

Brigham, E. O. [1988]. *The Fast Fourier Transform and its Applications*, Prentice Hall, Upper Saddle River, NJ.

Bribiesca, E. [1981]. "Arithmetic Operations Among Shapes Using Shape Numbers," *Pattern Recog.*, vol. 13, no. 2, pp. 123–138.

Bribiesca, E., and Guzman, A. [1980]. "How to Describe Pure Form and How to Measure Differences in Shape Using Shape Numbers," *Pattern Recog.*, vol. 12, no. 2, pp. 101–112.

Canny, J. [1986]. "A Computational Approach for Edge Detection," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 8, no. 6, pp. 679–698.

Dempster, A. P., Laird, N. M., and Ruben, D. B. [1977]. "Maximum Likelihood from Incomplete Data via the EM Algorithm," *J. R. Stat. Soc. B*, vol. 39, pp. 1–37.

Di Zenzo, S. [1986]. "A Note on the Gradient of a Multi-Image," *Computer Vision, Graphics and Image Processing*, vol. 33, pp. 116–125.

Floyd, R. W. and Steinberg, L. [1975]. "An Adaptive Algorithm for Spatial Gray Scale," *International Symposium Digest of Technical Papers*, Society for Information Displays, 1975, p. 36.

Gardner, M. [1970]. "Mathematical Games: The Fantastic Combinations of John Conway's New Solitaire Game 'Life', " *Scientific American*, October, pp. 120–123.

Gardner, M. [1971]. "Mathematical Games On Cellular Automata, Self-Reproduction, the Garden of Eden, and the Game 'Life', " *Scientific American*, February, pp. 112–117.

Hanisch, R. J., White, R. L., and Gilliland, R. L. [1997]. "Deconvolution of Hubble Space Telescope Images and Spectra," in *Deconvolution of Images and Spectra*, P.A. Jansson, ed., Academic Press, NY, pp. 310–360.

Haralick, R. M. and Shapiro, L. G. [1992]. *Computer and Robot Vision*, vols. 1 & 2, Addison-Wesley, Reading, MA.

索　引

E

F

［Ｇｅｎｅｒａｌ Ｉｎｆｏｒｍａｔｉｏｎ］
书名＝数字图像处理 （ＭＡＴＬＡＢ版）
作者＝
页数＝４７３
ＳＳ号＝０
出版日期＝